UFR Series NFC reader API (Application Programming Interface)

uFR series of readers operates with Mifare® Classic and other RFID contactless cards and tags which communication interface is compliant to ISO / IEC 14443 standard, including NFC tags.

Since contactless RFID cards have a lot of specifics that are not supported by any standard Windows or other platform specific API, uFR Series readers are supplied with separate API interface placed in the dynamic library called uFCoder. Depending on platform and architecture, there are prefixes and suffixes to this name and also file extension.

Standard library package contains libraries for Windows x86 and x64 architecture (DLL's), Linux x86 and x64 (.so files), ARM and ARM HF (.so files), Mac OSX (.dylib files), header file (.h) and libraries for static linking (.a and .lib files).

Alternatively this reader can communicate over a virtual COM port using appropriate available VCOM protocol. In this way, uFR Series readers can be used on any platform for which there is still no direct software support but there is existance of serial interface.

API specification (applies also to the VCOM protocol) contains functions that:

- emulate linear address space on the MIFARE® cards,
- directly addressing blocks on the MIFARE® card - the block address mode,
- indirectly addressing blocks on the MIFARE® cards, combining sector and blocks addresses within the sector - the sector address mode.

This way of data addressing is performed in accordance with the manufacturer's documentation for addressing the MIFARE® card.

Allows four methods of authentication for card data access:

- "**Reader key authentication" -** the default authentication mode. For this mode the keys are stored into the reader (with a maximum of 32 key with indexes from 0 to 31) and the key index is sent with related functions. In the case of functions that emulate linear address space in this method of authentication, the use of the same key for all sectors (or at least for those who are in default range for linear addressing) is default.

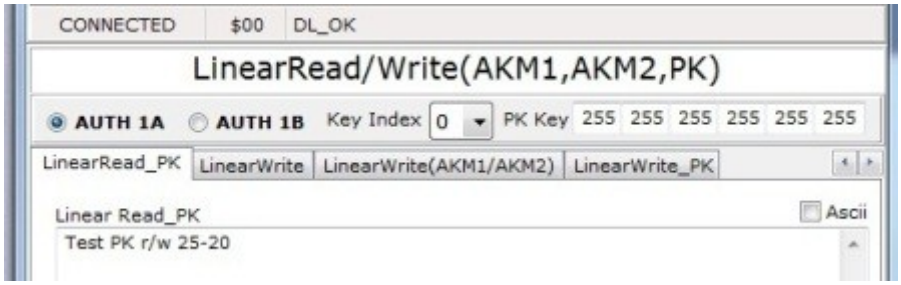
- "**Automatic key mode 1" (AKM1) and "Automatic key mode 2" (AKM2)** are optional, automatic modes of authentication. This modes enable automatic selection of keys stored in the reader on the basis of the block address or a combination of block and sector address within the sector. These modes could be used in emulation of a linear address space because after the address conversion in the readers software it is performed automatic keys selection for the authentication of a block or sector. The difference between AKM1 and AKM2 is only in the way of automatic selection of A and B keys performing.

When using **AKM1** mode it is accepted that the index keys in the reader from 0 to 15 are appropriate with A sectors keys from 0 to 15 and the index keys in the reader from 16 to 31 are appropriate with B sectors keys from 0 to 15.

When used **AKM2** mode, even key indexes in the reader (0, 2, ..., 28, 30) are accepted as a sectors keys from 0 to 15 respectively and the odd key indexes in the reader are accepted as B sectors keys from 0 to 15. This is certainly true for Mifare ® 1K.

For MIFARE ® card MINI only the first five keys A and B can be used (in AKM1 key index from 0 to 4 for A keys and from 16 to 20 for B keys, in AKM2 mode for A key index 0, 2, 4, 6 and 8 and for B keys 1, 3, 5, 7 and 9) because this cards contain only that much sectors.

On MIFARE® 4K there are 40 sectors so the lower and upper address space are organized into the 2K. With these cards AKM1 and AKM2 modes are organized in such a way that the same keys indexes from the reader corresponding sectors 0 to 15 and 16 to 32. For the last 8 sectors (sectors 32 to 39) the same readers keys are used that correspond to sectors 0 to 7 and 16 to 23.



The last method of authentication is **"Provided key" (PK)**. In this mode, you do not use keys stored in the reader but the keys are sent directly from the code with API functions. This mode does not provide any security, so its use is not recommended except under strictly controlled conditions or for testing purposes.

A special function for the sector trailer blocks card entry is implemented which is a very simplified calculation of the bytes values containing the access bits. This avoids the danger of permanently blocking the entire sectors of the card due to wrong bits format which controls access to blocks of a sector.

For those with more experience in working with MIFARE cards, the so-called unsafe option is left for the sector trailer blocks manipulation.

● There is a method for linear emulation mode, which formats the card sector trailer blocks in the same way ie. sets a unique keys and access bits for the entire card. This is a very simplified way of the card initialization for linear approach.

● API contains a set of functions for manipulating the cards value blocks. Four-byte values read and write are supported with a value blocks automatically formatted for the appropriate specification. The increment and decrement blocks value is also supported.

General functions for working with the reader

ReaderOpen: Opens a port of connected reader. In the case of multi-thread applications, developers must be careful to synchronize access to readers resources to avoid unforeseen situations.

ReaderOpenByType: Opens a port of connected reader by providing reader family type. Useful function for speed up opening for non uFR basic reader type, like uFR RS232 or XRC. Supported parameter's values:

- 0 – auto, same as call ReaderOpen()
- 1 : uFR type (1 Mbps)
- 2 : uFR RS232 type (115200 bps)
- 3 : XRC type (250 Kbps)

ReaderOpenEx: extended function for covering several situations of the reader opening.

Prototype:

```
UFR_STATUS ReaderOpenEx(uint32_t reader_type,
                        c_string port_name,
                        uint32_t port_interface,
                        void *arg);
```

Arguments:

- Supported values for argument **`reader_type`**:
  - 0 : auto mode – first try type 1, if no reader found - then type 2, then type 3
  - 1 : uFR type (1 Mbps)
  - 2 : uFR RS232 type (115200 bps)

- 3 : XRC type (250 Kbps)

- Argument **port_name** is c-string type. If provide port name function try to open port by given serial name. If provide NULL or empty string that is AUTO MODE and the ReaderOpenEx() will test all available ports on system.
- Argument **port_interface** define what kind of interface want to use while connecting to the printer. Supported value's:
  - 0 : auto mode – first try FTDI interfaces, and then try serial interfaces,
  - 1 : SERIAL – only try to open Reader via serial interface,
  - 2 : FTDI – only try to open Reader via FTDI interface
- The last argument **arg** is for testing and future purposes. Fix this argument to NULL.

GetReaderType: Returns the device type identifier. On uFR readers this value is 0xD1150021.

GetReaderSerialNumber: Returns the device serial number.

ReaderSoftRestart: Reader is restarted by software. This function sets all readers operating parameters to the default values and resets the close RF field, which practically resets all the cards in the field.

ReaderReset: Resets all the digital logic of readers hardware. This function can generally caled in the event that ReaderSoftRestart did not gave the desired results.

ReaderClose: Closes readers port. This enables access to the reader from other processes.

ReaderStillConnected: This function checks if the reader is still connected on system.

Functions declaration (C language):

```
UFR_STATUS ReaderStillConnected(unsigned int *connected);
```

- *connected* – Pointer to "unsigned int" type variable 32 bit long, where the information about availability of the reader is written. If the reader is connected on system and fully functioning than function store 1 (true) in that variable, otherwise, on some error – store zero.

ReaderKeyWrite: Sets the keys for authentication to the reader when reading and manipulating data on the cards. The keys are entered in a special readers area in EEPROM that can not be read anymore which provided protection against unauthorized access.

Function declaration (C language):

```
unsigned long ReaderKeyWrite(const unsigned char *aucKey,
                             unsigned char ucKeyIndex);
```

- *aucKey Pointer* to an array of 6 bytes containing the key. Key bytes can have any value in the range 0 to 255. The transport keys on the new cards should have all the bits degrades gracefully (all key bytes have a value of 255)
- *ucKeyIndex Index* in the reader where the user intends to store the new key. Possible values are 0 to 31.

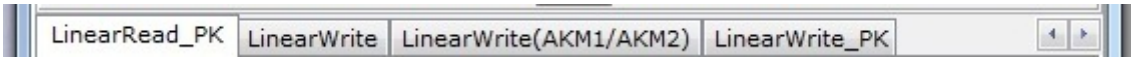
ReaderUISignal: The function is used to control the reader light and sound signal. There are four modes of light signals and five sound modes.

Function declaration (C language):

```
unsigned long ReaderUISignal(unsigned char ucLightSignalMode,
                             unsigned char ucBeepSignalMode);
```

- ucLightSignalMode Defines the light signals mode. It can have values from 0 to 4. A value of 0 indicates light signals inactivity.
- ucBeepSignalMode Defines the sound signals mode. It can have values from 0 to 5. A value of 0 indicates sound signals inactivity.

Functions for working with cards



By type of data they work with, the functions are classified in:

- Functions for manipulating card **data blocks**
- Functions for manipulating card **value blocks.**

According to the card data addressing method, this function are divided into:

- Functions that **emulate the linear address space**
- Functions that use **bloc addressing**
- Functions that use **sector addressing**

Functions for cards data manipulating are sorted according to the authentication method into the function sets recognizable by the suffix of the authentication method:

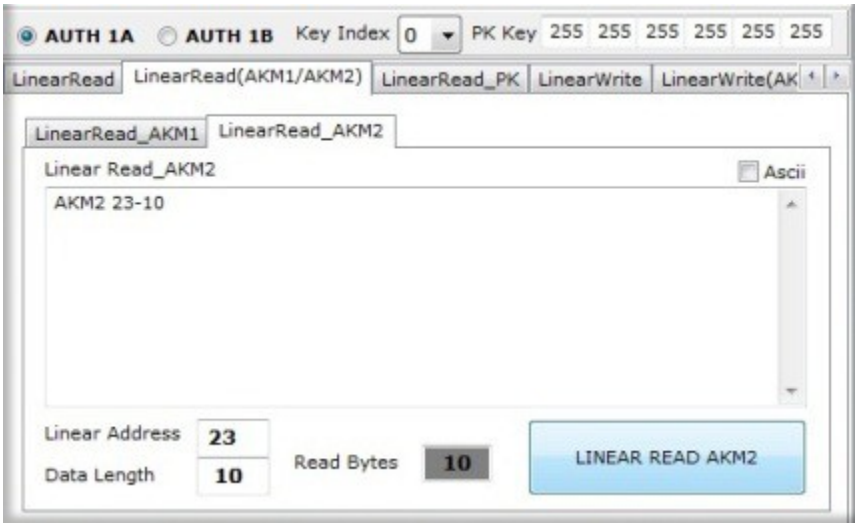
- *"Reader key authentication"* is the default authentication method so function of this group do not have any suffix
- Functions with the **_AKM1** suffix use *"Automatic key mode 1"*
- Functions with the **_AKM2** suffix use "Automatic key mode 2"

Functions with the **_PK** suffix use *"Provided key"* method.

General functions for working with cards

GetCardId: This function returns the type identifier and card serial number placed into the reader. Reader supports only cards that have 4 byte serial number (UID size: single) according to the standard ISO / IEC 14443 A.

Functions that emulate the linear address space



- ***LinearRead***
- ***LinearRead_AKM1***
- ***LinearRead_AKM2***
- ***LinearRead_PK***

Functions declaration (C language):

```
unsigned long LinearRead(unsigned char *aucData,
                  unsigned short usLinearAddress,
                  unsigned short usDataLength,
                  unsigned short *lpusBytesReturned,
                  unsigned char ucAuthMode,
                  unsigned char ucReaderKeyIndex);

unsigned long LinearRead_AKM1(unsigned char *aucData,
                           unsigned short usLinearAddress,
                           unsigned short usDataLength,
                           unsigned short *lpusBytesReturned,
                           unsigned char ucAuthMode);

unsigned long LinearRead_AKM2(unsigned char *aucData,
                         unsigned short usLinearAddress,
                         unsigned short usDataLength,
                         unsigned short *lpusBytesReturned,
                         unsigned char ucAuthMode);

unsigned long LinearRead_PK(unsigned char *aucData,
                         unsigned short usLinearAddress,
                         unsigned short usDataLength,
                         unsigned short *lpusBytesReturned,
```

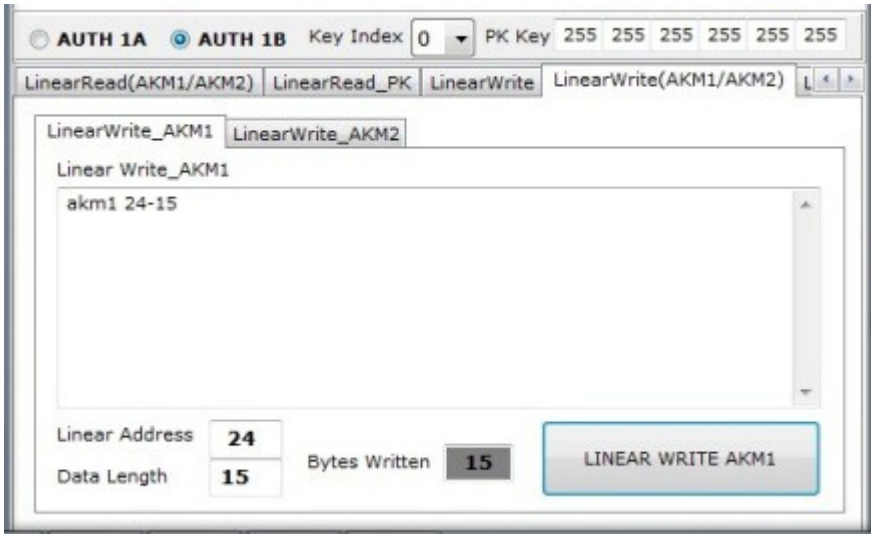
```
                                unsigned char ucAuthMode,
                                unsigned char *aucProvidedKey);
```

These functions are used for card data reading by using the linear address space emulation. The method for proving authenticity is determined by the suffix in the functions names:

- *aucData* - Pointer to the sequence of bytes where read data will be stored.
- *usLinearAddress* - Linear address on the card from which the data want to read
- *usDataLength* - Number of bytes for reading. For aucData a minimum usDataLength bytes must be allocated before calling the function
- *lpusBytesReturned* - Pointer to "unsigned short" type variable, where the number of successfully read bytes from the card is written. If the reading is fully managed this data is equal to the usDataLength parameter. If there is an error reading some of the blocks, the function returns all successfully read data in the aucData before the errors occurrence and the number of successfully read bytes is returned via this parameter

- *ucAuthMode* - This parameter **for Mifare Classic** tags defines whether to perform authentication with key A or key B. It can have two values, namely: AUTHENT1A (0x60) or AUTHENT1B (0x61). **For NTAG 21x, Ultralight EV1 and other T2T tags supporting PWD_AUTH** value 0x61 means "***use PWD_AUTH***" with LinearRead() or LinearRead_PK() functions. Value 0x60 with LinearRead() or LinearRead_PK() functions means "***without PWD_AUTH***" and in that case *you can send for ucReaderKeyIndex* or *aucProvidedKey parameters anything you want without influence on the result. For NTAG 21x, Ultralight EV1 and other T2T tags supporting PWD_AUTH you can use _AKM1 or _AKM2 function variants only* ***without PWD_AUTH*** *in any case of the valid values (0x60 or 0x61) provided for this parameter.*
  - ucReaderKeyIndex - The default method of authentication (when the functions without a suffix is used) performs the authenticity proving by using the selected key index from the reader. In the linear address mode, this applies to all sectors that are read
  - *aucProvidedKey* - Pointer to the six-byte string containing the key for authenticity proving in the "Provided Key" method. _PK Suffix in the name of the function indicates this method usage.

LinearRead group of functions in uFR firmware utilise FAST_READ ISO 14443-3 command with NTAG21x and Mifare Ultralight EV1 tags.



- ***LinearWrite,***
- ***LinearWrite_AKM1,***
- ***LinearWrite_AKM2,***
- ***LinearWrite_PK***

Functions declaration (C language):

```
unsigned long LinearWrite(const unsigned char *aucData,
                          unsigned short usLinearAddress,
                          unsigned short usDataLength,
                          unsigned short *lpusBytesWritten,
                          unsigned char ucAuthMode,
                          unsigned char ucReaderKeyIndex);

unsigned long LinearWrite_AKM1(const unsigned char *aucData,
                               unsigned short usLinearAddress,
                               unsigned short usDataLength,
                               unsigned short *lpusBytesWritten,
                               unsigned char ucAuthKey);

unsigned long LinearWrite_AKM2(const unsigned char *aucData,
                               unsigned short usLinearAddress,
                               unsigned short usDataLength,
                               unsigned short *lpusBytesWritten,
                               unsigned char ucAuthKey);
```

```
unsigned long LinearWrite_PK(const unsigned char *aucData,
                             unsigned short usLinearAddress,
                             unsigned short usDataLength,
                             unsigned short *lpusBytesWritten,
                             unsigned char ucAuthKey,
                             unsigned char *aucProvidedKey);
```

These functions are for writing data to the card using the emulation of linear address space. The method for proving authenticity is determined by the suffix in the functions names:

- *aucData* - Pointer to the sequence of bytes containing data for writing on the card
- *usLinearAddress* - Linear address of the card where the data writing is intend
- *usDataLength* - - Number of bytes for the entry. In aucData a minimum usDataLength bytes must be allocated before calling the function
- *lpusBytesWritten* - Pointer to a "unsigned short" type variable, where the number of successfully read bytes from the card is written. If the entry is a successfully completed this data is equal to the usDataLength parameter. If there was an error in writing some of the blocks, the function returns the number of successfully written bytes over this parameter
- *ucAuthKey* - This parameter ***for Mifare Classic*** *tags defines whether to perform authentication with key A or key B. It can have two values, namely: AUTHENT1A (0x60) or AUTHENT1B (0x61).* ***For NTAG 21x, Ultralight EV1 and other T2T tags supporting PWD_AUTH*** *value 0x61 means "****use PWD_AUTH****" with LinearWrite() or LinearWrite_PK() functions. Value 0x60 with LinearWrite() or LinearWrite_PK() functions means "****without PWD_AUTH****" and in that case you can send for ucReaderKeyIndex or aucProvidedKey parameters anything you want without influence on the result. For NTAG 21x, Ultralight EV1 and other T2T tags supporting PWD_AUTH you can use _AKM1 or _AKM2 function variants only* ***without PWD_AUTH*** *in any case of the valid values (0x60 or 0x61) provided for this parameter.*
  - *ucReaderKeyIndex* - The default method of authentication (when the functions without a suffix is used) performs the authenticity proving by using the selected key index from the reader. In the linear address mode, this applies to all sectors that are written
  - aucProvidedKey - Pointer to the sixth byte string containing the key for authenticity proving in the "Provided Key" method. _PK Suffix in the name of the function indicates this method usage.

- **LinearFormatCard,**
- **LinearFormatCard_AKM1,**
- **LinearFormatCard_AKM2,**

- **LinearFormatCard_PK**

Functions description:

These functions are used for new keys A and B writing as well as access bits in the trailers of all card sectors. The setting of ninth trailers bytes is enabled (a general-purpose byte where any value can be entered). In all the card sector trailers the same value is set for the entire card so the same keys and access rights are valid. As it is necessary to prove the authenticity on the base of previous keys before writing into the sector trailers, these functions are potentially suitable to initialize the new card (the authentication is performed with transportation keys, all the key bytes are 0xFF) or to re-initialize the card with the same keys and access rights for all sectors. Certainly, there must always be careful about the previously set access rights (access bits) on the cards in case the changing of some keys or bits for access rights control is disabled.

Function declaration (C language):

```
unsigned long LinearFormatCard(const unsigned char *aucNewKeyA,
                               unsigned char ucBlocksAccessBits,
                               unsigned char ucSectorTrailersAccessBits,
                               unsigned char ucSectorTrailersByte9,
                               const unsigned char *aucNewKeyB,
                               unsigned char *lpucSectorsFormatted,
                               unsigned char ucAuthMode,
                               unsigned char ucReaderKeyIndex);

unsigned long
LinearFormatCard_AKM1(const unsigned char *aucNewKeyA,
                      unsigned char ucBlocksAccessBits,
                      unsigned char ucSectorTrailersAccessBits,
                      unsigned char ucSectorTrailersByte9,
                      const unsigned char *aucNewKeyB,
                      unsigned char *lpucSectorsFormatted,
                      unsigned char ucAuthMode);

unaigned long
LinearFormatCard_AKM2(const unsigned char *aucNewKeyA,
                      unsigned char ucBlocksAccessBits,
                      unsigned char ucSectorTrailersAccessBits,
                      unsigned char ucSectorTrailersByte9,
                      const unsigned char *aucNewKeyB,
                      unsigned char *lpucSectorsFormatted,
                      unsigned char ucAuthMode);
```

```
unsigned long
LinearFormatCard(const unsigned char *aucNewKeyA,
                 unsigned char ucBlocksAccessBits,
                 unsigned char ucSectorTrailersAccessBits,
                 unsigned char ucSectorTrailersByte9,
                 const unsigned char *aucNewKeyB,
                 unsigned char *lpucSectorsFormatted,
                 unsigned char ucAuthMode,
                 unsigned char *aucProvidedKey);
```

Greater flexibility in sector trailers initiating is offered by **SectorTrailerWrite** functions group :

- *aucNewKeyA* - Pointer on 6 bytes array containing a new A key
- *ucBlocksAccessBits* - The access bits values that define permissions for all data blocks on the card. It can have values 0 to 7
- *ucSectorTrailersAccessBits* - The access bits value that define access permissions for all the card sector trailers. It can have values 0 to 7
- *ucSectorTrailersByte9* - The ninth byte value of all card sectors trailers. It can contain any value
- *aucNewKeyB* - *P*ointer on 6 bytes array containing a new B key
- *lpucSectorsFormatted* - Pointer to a "unsigned char" type variable through which the number of successfully formatted sectors trailers returns. Eg. if all the sectors trailers are successfully initialized, on the MIFARE® 1K, through this parameter it returns the value 16 which represents the number of sectors on this card. In case of error the parameter is an indication of the number of successfully initialized sectors starting from zero.
- *ucAuthMode* - This parameter defines whether to perform authentication A key or B key. It can have two values, namely: AUTHENT1A (0x60) or AUTHENT1B (0x61).
- *ucReaderKeyIndex* - The default method of authentication (when the functions without a suffix is used) performs the authenticity proving by using the selected key index from the reader. In the linear address mode, this applies to all sectors that are written

aucProvidedKey - Pointer to the sixth byte string containing the key for authenticity proving in the "Provided Key" method. _PK Suffix in the name of the function indicates this method usage.

Functions for working with data blocks

- **BlockRead,**
- **BlockRead_AKM1,**
- **BlockRead_AKM2,**
- **BlockRead_PK,**

Functions description:

This functions group is used for card block content reading. Always reads the entire block (16 bytes of the block). Functions use the so-called bloc addressing (the first card block has the address 0; first sector trailer has address 3, the next one 7, etc. until the last Mifare ® 1K block which is also a trailer of the last sector, has an address 63). These functions also allows reading of the sector trailers contents (its available part for reading, depending on the access rights set).

Functions declaration (C language):

```
unsigned long BlockRead(unsigned char *aucData,
                        unsigned char ucBlockAddress,
                        unsigned char ucAuthMode,
                        unsigned char ucKeyIndex);

unsigned long BlockRead_AKM1(unsigned char *aucData,
                             unsigned char ucBlockAddress,
                             unsigned char ucAuthMode);

unsigned long BlockRead_AKM2(unsigned char *aucData,
                             unsigned char ucBlockAddress,
                             unsigned char ucAuthMode);

 unsigned long BlockRead_PK(unsigned char *aucData,
                            unsigned char block_address,
                            unsigned char ucAuthMode,
                            const unsigned char *aucProvidedKey);
```

- *aucData* - Pointer to the number of bytes where read data will be stored. Must be allocated at least 16 bytes before calling the function.

- ucBlockAddress – block address.
- ucAuthMode – This parameter **for Mifare Classic** tags defines whether to perform authentication with key A or key B. It can have two values, namely: AUTHENT1A (0x60) or AUTHENT1B (0x61). **For NTAG 21x, Ultralight EV1 and other T2T tags supporting PWD_AUTH** value 0x61 means "***use PWD_AUTH***" with BlockRead() or BlockRead_PK() functions. Value 0x60 with BlockRead() or BlockRead_PK() functions means "***without PWD_AUTH***" and in that case *you can send for ucReaderKeyIndex* or *aucProvidedKey parameters anything you want without influence on the result. For NTAG 21x, Ultralight EV1 and other T2T tags supporting PWD_AUTH you can use _AKM1 or _AKM2 function variants only* ***without PWD_AUTH*** *in any case of the valid values (0x60 or 0x61) provided for this parameter.*

  - *UcReaderKeyIndex* – The default method of authentication (when the functions without a suffix is used) performs the authenticity proving by using the selected key index from the reader. In the linear address mode, this applies to all sectors that are written

- *aucProvidedKey* - Pointer to the sixth byte array containing the key for authenticity proving in the "Provided Key" method. _PK Suffix in the name of the function indicates this method usage.

These functions work the same as BlockRead group functions and are made for card block content reading. The only difference is that the sectoral addressing is used. That includes separately sending sector addresses and block addresses within a sector. For MIFARE® 1K card sector address may be in the range 0 to 15, and blocks address within the sector ranging from 0 to 3. For MIFARE ® 4k sector address may be in the range of 0 to 39 and since the second half of the address space organization is different (above 2 MB) blocks address in the last 8 sectors (sectors 32 to 39) may be in the range of 0 to 15. The entire block (16-byte block) is always read.

These functions can read the sector trailers contents (its available part for reading, depending on the access rights set).

- *aucData* - Pointer to the bytes array where read data are going to be stored. At least 16 bytes must be allocated before the function is called
- *ucSectorAddress* - Sector Address
- *ucBlockInSectorAddress* - Block address within a sector
- *ucAuthMode* - This parameter defines whether to perform authentication with A key or B key. It can have two values, namely: AUTHENT1A (0x60) or AUTHENT1B (0x61).
- *ucReaderKeyIndex* - The default method of authentication (when the functions without a suffix is used) performs the authenticity proving by using the selected key index from the reader. In the linear address mode, this applies to all sectors that are written
- *aucProvidedKey* - Pointer to the sixth byte array containing the key for authenticity proving in the "Provided Key" method. _PK Suffix in the name of the function indicates this method usage.
- **BlockWrite,**
- **BlockWrite_AKM1,**
- **BlockWrite_AKM2,**
- **BlockWrite_PK**

Functions description:

These functions are used for data entry (16 bytes at a time) into the card blocks. Functions use the so-called bloc addressing (the first card block has the address 0; first sector trailer has address 3, the next one 7, etc. until the last Mifare ® 1K block which is also a trailer of the last sector, has an address 63). This functions group don't allow direct data enter into the sector trailers. To do so, use the special functions SectorTrailerWrite and SectorTrailerWriteUnsafe.

Functions declaration (C language):

```
unsigned long BlockWrite(const unsigned char *aucData,
                         unsigned char ucBlockAddress,
                         unsigned char ucAuthMode,
                         unsigned char ucKeyIndex);

unsigned long BlockWrite_AKM1(const unsigned char *aucData,
                              unsigned char ucBlockAddress,
                              unsigned char ucAuthMode);

unsigned long BlockWrite_AKM2(const unsigned char *aucData,
                              unsigned char ucBlockAddress,
                              unsigned char ucAuthMode);

unsigned long BlockWrite_PK(const unsigned char *aucData,
                            unsigned char ucBlockAddress,
                            unsigned char ucAuthMode,
                            const unsigned char *aucProvidedKey);
```

- *aucData* - Pointer to the number of bytes where read data will be stored. Must be allocated at least 16 bytes before calling the function
- *ucBlockAddress* – Cards block address
- *ucAuthMode* – This parameter ***for Mifare Classic*** *tags defines whether to perform authentication with key A or key B. It can have two values, namely: AUTHENT1A (0x60) or AUTHENT1B (0x61).* ***For NTAG 21x, Ultralight EV1 and other T2T tags supporting PWD_AUTH*** *value 0x61 means "****use PWD_AUTH****" with BlockWrite() or BlockWrite_PK() functions. Value 0x60 with BlockWrite() or BlockWrite_PK() functions means "****without PWD_AUTH****" and in that case you can send for ucReaderKeyIndex or aucProvidedKey parameters anything you want without influence on the result. For NTAG 21x, Ultralight EV1 and other T2T tags supporting PWD_AUTH you can use _AKM1 or _AKM2 function variants only* ***without PWD_AUTH*** *in any case of the valid values (0x60 or 0x61) provided for this parameter.*
  - *ucReaderKeyIndex* - The default method of authentication (when the functions without a suffix is used) performs the authenticity proving by using the selected key index from the reader. In the linear address mode, this applies to all sectors that are read
  - *aucProvidedKey* - Pointer to the sixth byte array containing the key for authenticity proving in the "Provided Key" method. _PK Suffix in the name of the function indicates this method usage.

FORBIDEN_DIRECT_WRITE_IN_SECTOR_TRAILER.

- **BlockInSectorWrite,**
- **BlockInSectorWrite_AKM1,**
- **BlockInSectorWrite_AKM2,**
- **BlockInSectorWrite_PK**

Function description:

These functions work the same as BlockWrite group functions, they are used for data entry (16 bytes at a time) into card blocks. The only difference is the use of sector addressing. Sector addressing means separate sending sector and block addresses within a sector. For MIFARE® 1K card sector address may be in the range 0 to 15, and blocks address within the sector ranging from 0 to 3. For MIFARE ® 4k sector address may be in the range of 0 to 39 and since the second half of the address space organization is different (above 2 MB) blocks address in the last 8 sectors (sectors 32 to 39) may be in the range of 0 to 15. This functions group don't allow direct data enter into the sector trailers. To do so, use the special functions SectorTrailerWrite and SectorTrailerWriteUnsafe

Function declaration (C language)

```
unsigned long BlockWrite(const unsigned char *aucData,
                         unsigned char ucSectorAddress,
                         unsigned char ucBlockInSectorAddress,
                         unsigned char ucAuthMode,
                         unsigned char ucKeyIndex);

unsigned long BlockWrite_AKM1(const unsigned char *aucData,
                              unsigned char ucSectorAddress,
                              unsigned char ucBlockInSectorAddress,
                              unsigned char ucAuthMode);

unsigned long BlockWrite_AKM2(const unsigned char *aucData,
                              unsigned char ucSectorAddress,
                              unsigned char ucBlockInSectorAddress,
                              unsigned char ucAuthMode);

unsigned long BlockWrite_PK(const unsigned char *aucData,
                            unsigned char ucSectorAddress,
                            unsigned char ucBlockInSectorAddress,
                            unsigned char ucAuthMode,
                            const unsigned char *aucProvidedKey);
```

- *aucData* - Pointer to the number of bytes where read data will be stored. Must be allocated at least 16 bytes before calling the function
- *ucSectorAddress* - Sector address
- *ucBlockInSectorAddress* - Block address in the sector
- *ucAuthMode* - This parameter defines whether to perform authentication with A key or B key. It can have two values, namely: AUTHENT1A (0x60) or AUTHENT1B (0x61)
- *ucReaderKeyIndex* - The default method of authentication (when the functions without a suffix is used) performs the authenticity proving by using the selected key index from the reader. In the linear address mode, this applies to all sectors that are written

- *aucProvidedKey* - Pointer to the sixth byte array containing the key for authenticity proving in the "Provided Key" method. _PK Suffix in the name of the function indicates this method usage.

FORBIDEN_DIRECT_WRITE_IN_SECTOR_TRAILER.

- **SectorTrailerWrite,**
- **SectorTrailerWrite_AKM1,**
- **SectorTrailerWrite_AKM2,**
- **SectorTrailerWrite_PK**

Functions description:

These functions are used for data writing in the card sector trailers. Functions can also be used for sector trailers block addressing as well as for the sector addressing which is determined by the ucAddressingMode parameter. In the case of block addressing, the first card block has the address 0; trailer has a first sector address 3 and the next 7, etc. until the last block of Mifare® 1k which is also a trailer of the last sector and has an address 63. This group of functions simplifies the bits manipulation for blocks access rights setting (access bits) and minimizes the possibility of permanent blocking of the whole sector due to incorrect formatting of these bits. Formatting the access bits is made by the reader before the writing. API users can choose the appropriate blocks access rights which are represented by values 0 to 7 and to transmit them to these functions.

For sector trailers the following access rights are valid:

Access bits			Access values (forwarded to the function)	Access rights					
				A key		Bytes containing access bits and 9 byte		B Key	
C1	C2	C3		Read	Write	Read	Write	Read	Write
0	0	0	0	forbiden	A Key	A Key	forbiden	A Key	A Key
0	0	1	1	forbiden	A Key	A Key	A Key	A Key	A Key
0	1	0	2	forbiden	forbiden	A Key	forbiden	A Key	forbiden
0	1	1	3	forbiden	B Key	A or B Key	forbiden	forbiden	B Key
1	0	0	4	forbiden	B Key	A or B Key	forbiden	forbiden	B Key
1	0	1	5	forbiden	forbiden	A or B Key	forbiden	forbiden	forbiden
1	1	0	6	forbiden	forbiden	A or B Key	forbiden	forbiden	forbiden
1	1	1	7	forbiden	forbiden	A or B Key	forbiden	forbiden	forbiden

Table 1: Access rights for the sector trailers

For sector trailers following access rights are valid:

- Access bits C1 C2 C3
- Access values (submitted to the function)

- Access rights
- Key A bytes containing the access bits and the nine byte key B
- Reading and writing

For blocks the following access rights are valid:

Access bits			Access values (forwarded to the function)	Access rights			
C1	C2	C3		Read	Write	Increment	Decrement
0	0	0	0	A or B Key*	A or B Key*	A or B Key*	A or B Key*
0	0	1	1	A or B Key*	forbidden	forbiden	A or B Key*
0	1	0	2	A or B Key*	forbidden	forbiden	forbiden
0	1	1	3	B Key*	B Key*	forbiden	forbiden
1	0	0	4	A or B Key*	B Key*	forbiden	forbiden
1	0	1	5	B Key*	forbiden	forbiden	forbiden
1	1	0	6	A or B Key*	B Key*	B Key*	A or B Key*
1	1	1	7	forbiden	forbiden	forbiden	forbiden

Table 2: Access rights for the blosks

*) If the access rights for the sector trailer of an appropriate sector set up so that it is possible to readB Key, it can not be used for authentication in any of the cases. These functions also sets new sector keys if it is permitted to access rights.

For blocks the fallowing access rights are valid:

- Access bits C1 C2 C3
- Access values (submitted to the function)
- Access rights
- Reading, writing, increment, decrement

Functions declaration (C language):

```
unsigned long
SectorTrailerWrite(unsigned char ucAddressingMode,
                   unsigned char ucAddress,
                   const unsigned char *aucNewKeyA,
                   unsigned char ucBlock0AccessBits,
                   unsigned char ucBlock1AccessBits,
                   unsigned char ucBlock2AccessBits,
                   unsigned char ucSectorTrailerAccessBits,
                   unsigned char ucSectorTrailerByte9,
                   const unsigned char *aucNewKeyB,
                   unsigned char ucAauthMode,
                   unsigned char ucKeyIndex);

unsigned long
SectorTrailerWrite_AKM1(unsigned char ucAddressingMode,
                        unsigned char ucAddress,
                        const unsigned char *aucNewKeyA,
                        unsigned char ucBlock0AccessBits,
                        unsigned char ucBlock1AccessBits,
                        unsigned char ucBlock2AccessBits,
                        unsigned char ucSectorTrailerAccessBits,
                        unsigned char ucSectorTrailerByte9,
                        const unsigned char *aucNewKeyB,
                        unsigned char ucAauthMode);

unsigned long
SectorTrailerWrite_AKM2(unsigned char ucAddressingMode,
                        unsigned char ucAddress,
                        const unsigned char *aucNewKeyA,
                        unsigned char ucBlock0AccessBits,
                        unsigned char ucBlock1AccessBits,
                        unsigned char ucBlock2AccessBits,
                        unsigned char ucSectorTrailerAccessBits,
                        unsigned char ucSectorTrailerByte9,
                        const unsigned char *aucNewKeyB,
                        unsigned char ucAauthMode);
```

```
unsigned long
SectorTrailerWrite_PK(unsigned char ucAddressingMode,
                      unsigned char ucAddress,
                      const unsigned char *aucNewKeyA,
                      unsigned char ucBlock0AccessBits,
                      unsigned char ucBlock1AccessBits,
                      unsigned char ucBlock2AccessBits,
                      unsigned char ucSectorTrailerAccessBits,
                      unsigned char ucSectorTrailerByte9,
                      const unsigned char *aucNewKeyB,
                      unsigned char ucAauthMode,,
                      const unsigned char *aucProvidedKey);
```

- *ucAddressingMode* - Specifies the address mode. Possible values of this parameter are BLOCK_ADDRESS_MODE (0x00) or SECTOR_ADDRESS_MODE (0x01). If any other value is sent the function returns an error code WRONG_ADDRESS_MODE
- *ucAddress* - Sectors or sector trailers blocks address, depending on ucAddressingMode. When using a sector-address mode, then, for instance, the MIFARE Classic 1K card, the range can be from 0 to 15 (this card has 16 sectors). The same card type in the block addressing mode can use the values from 0 to 63 provided that an error occurs if the addressed block is not also the sector trailer.
- *aucNewKeyA* - Pointer to the 6 byte array that represents a new A key for a specified sector which will be set if that is previously allowed with the access rights
- *aucNewKeyB* - Pointer to the six-byte array that represents a new B key for a specified sector which will be set if that is previously allowed with the access rights
- *ucBlock0AccessBits* - Access value for the 0 block of a sector.

MIFARE ® 4k has a different organization for the last 8 sectors, the second half of the address space. Therefore, in these sectors the access rights are set as follows:

- access rights to the first 5 blocks - ucBlock1AccessBits Access value block for the first sector
- prava pristupa drugih 5 blokova - ucBlock2AccessBits Access value block for the first sector
- access rights to the last 5 blocks:
  -
  - *ucSectorTrailerAccessBits* - Access value for a sector trailer
  - *ucSectorTrailerByte9* - The ninth sector trailers byte is a byte for general purpose where any single-byte value can be entered

- *ucAuthMode* - This parameter defines whether to perform authentication with key A or key B. It can have two values, namely: AUTHENT1A (0x60) or AUTHENT1B (0x61)
- *ucReaderKeyIndex* - The default method of authentication (when the functions without a suffix is used) performs the authenticity proving by using the selected key index from the reader. In the linear address mode, this applies to all sectors that are written
- *aucProvidedKey* - Pointer to the sixth byte array containing the key for authenticity proving in the "Provided Key" method. _PK Suffix in the name of the function indicates this method usage.

- **SectorTrailerWriteUnsafe,**
- **SectorTrailerWriteUnsafe_AKM1,**
- **SectorTrailerWriteUnsafe_AKM2,**
- **SectorTrailerWriteUnsafe_PK**

Functions description:

These functions have the same purpose as the function of the SectorTrailerWrite group with the difference in sending the "raw" sector trailers content and the errors are possible while formatting access bits values for entering. These functions are intended for developers with experience in working with MIFARE cards. All rules mentioned for the SectorTrailerWrite group functions applies to these functions, except the option of the "raw" data for sector trailer entry.

Functions declaration (C language):

```
unsigned long
SectorTrailerWriteUnsafe(unsigned char ucAddressingMode,
                         unsigned char ucAddress,
                         const unsigned char *aucSectorTrailer,
                         unsigned char ucAauthMode,
                         unsigned char key_index);

unsigned long
SectorTrailerWriteUnsafe_AKM1(unsigned char ucAddressingMode,
                              unsigned char ucAddress,
                          const unsigned char *aucSectorTrailer,
                              unsigned char ucAauthMode);

nsigned long
SectorTrailerWriteUnsafe_AKM2(unsigned char ucAddressingMode,
                              unsigned char ucAddress,
                        const unsigned char *aucSectorTrailer,
                              unsigned char ucAauthMode);
```

```
unsigned long
SectorTrailerWriteUnsafe_PK(unsigned char ucAddressingMode,
                            unsigned char ucAddress,
                      const unsigned char *aucSectorTrailer,
                            unsigned char ucAauthMode,
                        const unsigned char *aucProvidedKey);
```

- *ucAddressingMode* - Specifies the address mode. Possible values of this parameter are BLOCK_ADDRESS_MODE (0x00) or SECTOR_ADDRESS_MODE (0x01). If any other value is been sent the function returns an error code WRONG_ADDRESS_MODE.
- *ucAddress* - Sectors or sector trailers block address, depending on ucAddressingMode.

When using a sector address mode, then,in the case of MIFARE ® 1K card, the range can be from 0 to 15 (this card has 16 sectors) and the same card type in block addressing mode can use the values 0 to 63 with the possible error if the addressed block isn't also the sector trailer.

- *aucSectorTrailer* - Pointer to 6 byte array that contains the "raw" data for the address sector trailer entry
- *ucAuthMode* - This parameter defines whether to perform authentication key A or key B. It can have two values, namely: AUTHENT1A (0x60) or AUTHENT1B (0x61)
- *ucReaderKeyIndex* - The default method of authentication (when the functions without a suffix is used) performs the authenticity proving by using the selected key index from the reader. In the linear address mode, this applies to all sectors that are written

aucProvidedKey - Pointer to the sixth byte array containing the key for authenticity proving in the "Provided Key" method. _PK Suffix in the name of the function indicates this method usage.

Functions for working with value blocks

Value blocks represents an optional MIFARE® card functionality. This is actually a mode in which the entire block of data on the card (16 bytes) represents one four-byte value. In this mode, you can add any data block on the card (except of course, block 0, the zero sector and sector trailer). The values in the value blocks are formatted in a special way and in addition to value records contains the one byte address value, which gives users the added ability to implement the backup system.

D-Logic card readers takes care of the proper value blocks formatting so the set of functions that handle only with four byte values are available to users. It should be mentioned that the use of value blocks makes sense if the access rights to desired block are set on values 1, 6 or 0 (the default in new cards) which allows their values increment and decrement. First of all, value blocks must be initiated, value and associated address must be in compliance with the appropriate format of sixteen byte

records. The best and easiest way for value blocks initialization is with a set of API functions ValueBlockWrite or ValueBlockInSectorWrite.

- **ValueBlockRead,**
- **ValueBlockRead_AKM1,**
- **ValueBlockRead_AKM2,**
- **ValueBlockRead_PK**

Functions description:

These functions are used to read the fourth byte value of value blocks. In addition they are returning the associated address stored in the value block. Functions used block addressing (the first card block has the address 0; first sector trailer has address 3, the next one 7, etc. until the last Mifare ® 1K block which is also a trailer of the last sector, has an address 63)

Function declaration (C language):

```
unsigned long ValueBlockRead(long *lValue,
                             unsigned char *ucValueAddr,
                             unsigned char ucBlockAddress,
                             unsigned char ucAuthMode,
                             unsigned char ucKeyIndex);

unsigned long ValueBlockRead_AKM1(long *lValue,
                                  unsigned char *ucValueAddr,
                                  unsigned char ucBlockAddress,
                                  unsigned char ucAuthMode);

unsigned long ValueBlockRead_AKM1(long *lValue,
                                  unsigned char *ucValueAddr,
                                  unsigned char ucBlockAddress,
                                  unsigned char ucAuthMode);

unsigned long ValueBlockRead_PK(long *lValue,
                                unsigned char *ucValueAddr,
                                unsigned char ucBlockAddress,
                                unsigned char ucAuthMode,
                                const unsigned char *key);
```

- *lValue* - Pointer to a variable of a type "long" over which the block value returns
- *ucValueAddr* - Pointer to a variable of unsigned char type is returned via the one byte address which gives the added ability for a backup system implementation
- *ucBlockAddress* - Block address

- *ucAuthMode* - This parameter defines whether to perform authentication key A or key B. It can have two values, namely: AUTHENT1A (0x60) or AUTHENT1B (0x61).
- *ucReaderKeyIndex* - The default method of authentication (when the functions without a suffix is used) performs the authenticity proving by using the selected key index from the reader. In the linear address mode it applies to all sectors for writing
- *aucProvidedKey* - Pointer to the six-byte array that contains the key for authentication of the "Provided Key" method. _PK function name suffix indicates to the use of this method.
- **ValueBlockInSectorRead,**
- **ValueBlockInSectorRead_AKM1,**
- **ValueBlockInSectorRead_AKM2,**
- **ValueBlockInSectorRead_PK**

Functions description

These functions do the same as ValueBlockRead group functions and are proper for reading 4 byte values of the value blocks. In addition they return the associated address stored in the value block. The only difference is the use of so-called sectoral addressing. Sectoral addressing means separately sending sector and block addresses within a sector. For MIFARE® 1K card sector address may be in the range 0 to 15, and blocks address within the sector ranging from 0 to 3. For MIFARE ® 4k sector address may be in the range of 0 to 39 and since the second half of the address space organization is different (above 2 MB) blocks address in the last 8 sectors (sectors 32 to 39) may be in the range of 0 to 15.

Function declaration (C language):

```
unsigned long ValueBlockInSectorRead(long *lValue,
                                     unsigned char *ucValueAddr,
                                     unsigned char ucSectorAddress,
                                unsigned char ucBlockInSectorAddress,
                                     unsigned char ucAuthMode,
                                     unsigned char ucKeyIndex);

unsigned long ValueBlockInSectorRead_AKM1(long *lValue,
                                          unsigned char *ucValueAddr,
                                        unsigned char ucSectorAddress,
                                   unsigned char ucBlockInSectorAddress,
                                          unsigned char ucAuthMode);
```

```
unsigned long ValueBlockInSectorRead_AKM1(long *lValue,
                                          unsigned char *ucValueAddr,
                                        unsigned char ucSectorAddress,
                                    unsigned char ucBlockInSectorAddress,
                                          unsigned char ucAuthMode);

unsigned long ValueBlockInSectorRead_PK(long *lValue,
                                        unsigned char *ucValueAddr,
                                      unsigned char ucSectorAddress,
                                  unsigned char ucBlockInSectorAddress,
                                        unsigned char ucAuthMode,
                                  const unsigned char *aucProvidedKey);
```

- *lValue* - Pointer to a variable of a long type over which the value block returns
- *ucValueAddr* - Pointer to a variable of unsigned char type is returned via the one byte address which gives the added ability for a backup system implementation
- *ucSectorAddress* - Sector address
- *ucBlockInSectorAddress* - Block address in a sector
- *ucAuthMode* - This parameter defines whether to perform authentication key A or key B. It can have two values, namely: AUTHENT1A (0x60) or AUTHENT1B (0x61)
- *ucReaderKeyIndex* - e default method of authentication (when the functions without a suffix is used) performs the authenticity proving by using the selected key index from the reader. In the linear address mode it applies to all sectors for writing
- *aucProvidedKey* - Pointer to the six-byte array that contains the key for authentication of the "Provided Key" method. _PK function name suffix indicates to the use of this method.
- **ValueBlockWrite,**
- **ValueBlockWrite_AKM1,**
- **ValueBlockWrite_AKM2,**
- **ValueBlockWrite_PK**

Functions description:

These functions are used to initialize and write fourth byte value blocks values and store the associated address in the value block. Functions using the so-called block addressing (the first card block has the address 0; trailer has a first sector address 3 and the next 7, etc. until the last block of Mifare® 1k which is also a trailer of the last sector and has an address 63).

Function declaration (C language):

```
unsigned long ValueBlockWrite(long lValue,
                              unsigned char ucValueAddr,
                              unsigned char ucBlockAddress,
                              unsigned char ucAuthMode,
                              unsigned char ucKeyIndex);

unsigned long ValueBlockWrite_AKM1(long lValue,
                                   unsigned char ucValueAddr,
                                   unsigned char ucBlockAddress,
                                   unsigned char ucAuthMode);

unsigned long ValueBlockWrite_AKM2(long lValue,
                                   unsigned char ucValueAddr,
                                   unsigned char ucBlockAddress,
                                   unsigned char ucAuthMode);

unsigned long ValueBlockWrite_PK(long lValue,
                                 unsigned char ucValueAddr,
                                 unsigned char ucBlockAddress,
                                 unsigned char ucAuthMode,
                           const unsigned char *aucProvidedKey);
```

- *lValue* - Value for the value block entry
- *ucValueAddr* - Value block associated address
- *ucBlockAddress* - Block address
- *ucAuthMode* - This parameter defines whether to perform authentication with A key or B key. It can have two values, namely: AUTHENT1A (0x60) or AUTHENT1B (0x61).
- *ucReaderKeyIndex* - The default method of authentication (when the functions without a suffix is used) performs the authenticity proving by using the selected key index from the reader. In the linear address mode, this applies to all sectors that are written
- *aucProvidedKey* - Pointer to the sixth byte array containing the key for authenticity proving in the "Provided Key" method. _PK Suffix in the name of the function indicates this method usage.

- **ValueBlockInSectorWrite,**
- **ValueBlockInSectorWrite_AKM1,**
- **ValueBlockInSectorWrite_AKM2,**
- **ValueBlockInSectorWrite_PK**

Functions description:

These functions are similar to the ValueBlockWrite group functions. They use for entry, value blocks 4 bytes values initialization. In addition, stores the associated address into the block value. The only difference is the sectoral addressing usage. Sectoral addressing means separately sending sector and block addresses within a sector. For MIFARE® 1K card sector address may be in the range 0 to 15, and blocks address within the sector ranging from 0 to 3. For MIFARE ® 4k sector address may be in the range of 0 to 39 and since the second half of the address space organization is different (above 2 MB) blocks address in the last 8 sectors (sectors 32 to 39) may be in the range of 0 to 15.

Functions declaration (C language):

```
unsigned long ValueBlockInSectorWrite(long lValue,
                                      unsigned char ucValueAddr,
                                      unsigned char ucSectorAddress,
                                      unsigned char ucBlockInSectorAddress,
                                      unsigned char ucAuthMode,
                                      unsigned char ucKeyIndex);

unsigned long ValueBlockInSectorWrite_AKM1(long lValue,
                                           unsigned char ucValueAddr,
                                           unsigned char ucSectorAddress,
                                      unsigned char ucBlockInSectorAddress,
                                           unsigned char ucAuthMode);

unsigned long ValueBlockInSectorWrite_AKM2(long lValue,
                                           unsigned char ucValueAddr,
                                           unsigned char ucSectorAddress,
                                     unsigned char ucBlockInSectorAddress,
                                           unsigned char ucAuthMode);

 unsigned long ValueBlockInSectorWrite_PK(long lValue,
                                         unsigned char ucValueAddr,
                                         unsigned char ucSectorAddress,
                                  unsigned char ucBlockInSectorAddress,
                                              unsigned char ucAuthMode,
                                const unsigned char *aucProvidedKey);
```

- *lValue* - Values for the value block entry
- *ucValueAddr* - Value block associated address
- *ucSectorAddress* - Sector address
- *ucBlockInSectorAddress* - Block address of a sector
- *ucAuthMode* - This parameter defines whether to perform authentication key A or key B. It can have two values, namely: AUTHENT1A (0x60) or AUTHENT1B (0x61)

- *ucReaderKeyIndex* - The default method of authentication (when the functions without a suffix is used) performs the authenticity proving by using the selected key index from the reader. In the linear address mode it applies to all sectors for writing
- *aucProvidedKey* - Pointer to the six-byte array that contains the key for authentication of the "Provided Key" method. _PK function name suffix indicates to the use of this method.
- **ValueBlockIncrement,**
- **ValueBlockIncrement_AKM1,**
- **ValueBlockIncrement_AKM2,**
- **ValueBlockIncrement_PK**

Functions description:

This feature set is used to increment the value 4 byte value blocks. The value of value block increment is sent as a parameter of these functions. Functions use block addressing (the first card block has the address 0; first sector trailer has address 3, the next one 7, etc. until the last Mifare ® 1K block which is also a trailer of the last sector, has an address 63).

Funcution declaration (C language):

```
unsigned long
ValueBlockInSectorIncrement(long lIncrementValue,
                            unsigned char ucSectorAddress,
                            unsigned char ucBlockInSectorAddress,
                            unsigned char ucAuthMode,
                            unsigned char ucKeyIndex);

unsigned long
ValueBlockInSectorIncrement_AKM1(long lIncrementValue,
                                 unsigned char ucSectorAddress,
                           unsigned char ucBlockInSectorAddress,
                                   unsigned char ucAuthMode);

unsigned long
ValueBlockInSectorIncrement_AKM2(long lIncrementValue,
                                 unsigned char ucSectorAddress,
                           unsigned char ucBlockInSectorAddress,
                                 unsigned char ucAuthMode);
```

```
unsigned long
ValueBlockInSectorIncrement_PK(long lIncrementValue,
                               unsigned char ucSectorAddress,
                          unsigned char ucBlockInSectorAddress,
                                unsigned char ucAuthMode,
                           const unsigned char *aucProvidedKey);
```

- *lIncrementValue* - The value of value block increment
- *ucBlockAddress* - Block adress in a sector
- *ucAuthMode* - This parameter defines whether to perform authentication key A or key B. It can have two values, namely: AUTHENT1A (0x60) or AUTHENT1B (0x61)
- *ucReaderKeyIndex* - The default method of authentication (when the functions without a suffix is used) performs the authenticity proving by using the selected key index from the reader. In the linear address mode it applies to all sectors for writing
- *aucProvidedKey* - Pointer to the six-byte array that contains the key for authentication of the "Provided Key" method. _PK function name suffix indicates to the use of this method.

- **ValueBlockInSectorIncrement,**
- **ValueBlockInSectorIncrement_AKM1,**
- **ValueBlockInSectorIncrement_AKM2,**
- **ValueBlockInSectorIncrement_PK**

Functions description:

These functions has the same purpose as ValueBlockIncrement group functions and are used for reading 4 byte values of the value blocks. The value of value block increment is sent as a parameter of these functions. The only difference is the sectoral addressing usage. Sectoral addressing means separately sending sector and block addresses within a sector. For MIFARE® 1K card sector address may be in the range 0 to 15, and blocks address within the sector ranging from 0 to 3. For MIFARE ® 4k sector address may be in the range of 0 to 39 and since the second half of the address space organization is different (above 2 MB) blocks address in the last 8 sectors (sectors 32 to 39) may be in the range of 0 to 15.

Functions declaration (C language):

```
unsigned long
ValueBlockInSectorIncrement(long lIncrementValue,
                            unsigned char ucSectorAddress,
                            unsigned char ucBlockInSectorAddress,
                            unsigned char ucAuthMode,
                            unsigned char ucKeyIndex);
```

```
unsigned long
ValueBlockInSectorIncrement_AKM1(long lIncrementValue,
                                 unsigned char ucSectorAddress,
                               unsigned char ucBlockInSectorAddress,
                                   unsigned char ucAuthMode);

unsigned long
ValueBlockInSectorIncrement_AKM2(long lIncrementValue,
                                 unsigned char ucSectorAddress,
                               unsigned char ucBlockInSectorAddress,
                                 unsigned char ucAuthMode);

unsigned long
ValueBlockInSectorIncrement_PK(long lIncrementValue,
                                  unsigned char ucSectorAddress,
                             unsigned char ucBlockInSectorAddress,
                                    unsigned char ucAuthMode,
                             const unsigned char *aucProvidedKey);
```

- *lIncrementValue* - The value of value block increment
- *ucSectorAddress* - *S*ector address
- *ucBlockInSectorAddress* - Block address within a sector
- *ucAuthMode* - This parameter defines whether to perform authentication key A or key B. It can have two values, namely: AUTHENT1A (0x60) or AUTHENT1B (0x61)
- *ucReaderKeyIndex* - The default method of authentication (when the functions without a suffix is used) performs the authenticity proving by using the selected key index from the reader. In the linear address mode it applies to all sectors for writing
- *aucProvidedKey* - Pointer to the six-byte array that contains the key for authentication of the "Provided Key" method. _PK function name suffix indicates to the use of this method.

- **ValueBlockDecrement,**
- **ValueBlockDecrement_AKM1,**
- **ValueBlockDecrement_AKM2,**
- **ValueBlockDecrement_PK**

Functions description:

This set of functions is used to decrement 4 byte value of value blocks. The value of the value block decrement is sent as a parameter of these functions. Functions use block addressing (the first card block has the address 0; first sector trailer has address 3, the next one 7, etc. until the last Mifare ® 1K block which is also a trailer of the last sector, has an address 63).

Functions declaration (C language):

```
unsigned long ValueBlockDecrement(long lDecrementValue,
                                  unsigned char ucBlockAddress,
                                  unsigned char ucAuthMode,
                                  unsigned char ucKeyIndex);

unsigned long ValueBlockDecrement_AKM1(long lDecrementValue,
                                       unsigned char ucBlockAddress,
                                       unsigned char ucAuthMode);

unsigned long ValueBlockDecrement_AKM2(long lDecrementValue,
                                       unsigned char ucBlockAddress,
                                       unsigned char ucAuthMode);

unsigned long ValueBlockDecrement_PK(long lDecrementValue,
                                     unsigned char ucBlockAddress,
                                        unsigned char ucAuthMode,
                                  const unsigned char *aucProvidedKey);
```

- *lDecrementValue* - The value of value block decrement
- *ucBlockAddress* - Block address within a sector
- *ucAuthMode* - This parameter defines whether to perform authentication key A or key B. It can have two values, namely: AUTHENT1A (0x60) or AUTHENT1B (0x61)
- *ucReaderKeyIndex* - The default method of authentication (when the functions without a suffix is used) performs the authenticity proving by using the selected key index from the reader. In the linear address mode it applies to all sectors for writing
- *aucProvidedKey* - Pointer to the six-byte array that contains the key for authentication of the "Provided Key" method. _PK function name suffix indicates to the use of this method.

- **ValueBlockInSectorDecrement,**
- **ValueBlockInSectorDecrement_AKM1,**
- **ValueBlockInSectorDecrement_AKM2,**
- **ValueBlockInSectorDecrement_PK**

Functions description:

These functions work the same as ValueBlockDecrement group functions and are made for the value blocks 4 byte values decrement. The value of the value block decrement is sent as a parameter to these functions. Only difference is the sectoral addressing usage. That includes separately sending sector addresses and block addresses within a sector. For MIFARE® 1K card sector address may be in the range 0 to 15, and blocks address within the sector ranging from 0 to 3. For MIFARE ® 4k sector address may be in the range of 0 to 39 and since the second half of the

address space organization is different (above 2 MB) blocks address in the last 8 sectors (sectors 32 to 39) may be in the range of 0 to 15

Function declaration (C language):

```
unsigned long
ValueBlockInSectorDecrement(long lDecrementValue,
                            unsigned char ucSectorAddress,
                            unsigned char ucBlockInSectorAddress,
                            unsigned char ucAuthMode,
                            unsigned char ucKeyIndex);

unsigned long
ValueBlockInSectorDecrement_AKM1(long lDecrementValue,
                                 unsigned char ucSectorAddress,
                                unsigned char ucBlockInSectorAddress,
                                   unsigned char ucAuthMode);

unsigned long
ValueBlockInSectorDecrement_AKM2(long lDecrementValue,
                                 unsigned char ucSectorAddress,
                                unsigned char ucBlockInSectorAddress,
                                    unsigned char ucAuthMode);

unsigned long
ValueBlockInSectorDecrement_PK(long lDecrementValue,
                               unsigned char ucSectorAddress,
                           unsigned char ucBlockInSectorAddress,
                               unsigned char ucAuthMode,
                            const unsigned char *aucProvidedKey);
```

- *lDecrementValue* - The value of value block decrement
- *ucSectorAddress* - Sector address
- *ucBlockInSectorAddress* - Block address within a sector
- *ucAuthMode* - This parameter defines whether to perform authentication key A or key B. It can have two values, namely: AUTHENT1A (0x60) or AUTHENT1B (0x61)
- *ucReaderKeyIndex* - The default method of authentication (when the functions without a suffix is used) performs the authenticity proving by using the selected key index from the reader. In the linear address mode it applies to all sectors for writing

aucProvidedKey - Pointer to the six-byte array that contains the key for authentication of the "Provided Key" method. _PK function name suffix indicates to the use of this method.

Additional general functions for working with the cards

- **GetDlogicCardType**

Function description:

This is a function that return a value that corresponds to the type of the detected card. Values of the card type defined in the following list:

```
#define DL_MIFARE_ULTRALIGHT                0x01
#define DL_MIFARE_ULTRALIGHT_EV1_11         0x02
#define DL_MIFARE_ULTRALIGHT_EV1_21         0x03
#define DL_MIFARE_ULTRALIGHT_C              0x04
#define DL_NTAG_203                         0x05


#define DL_NTAG_210                         0x06
#define DL_NTAG_212                         0x07
#define DL_NTAG_213                         0x08
#define DL_NTAG_215                         0x09
#define DL_NTAG_216                         0x0A

#define DL_MIFARE_MINI                      0x20
#define DL_MIFARE_CLASSIC_1K                0x21
#define DL_MIFARE_CLASSIC_4K                0x22
#define DL_MIFARE_PLUS_S_2K           0x23
#define DL_MIFARE_PLUS_S_4K           0x24
#define DL_MIFARE_PLUS_X_2K           0x25
#define DL_MIFARE_PLUS_X_4K           0x26
#define DL_MIFARE_DESFIRE                   0x27
#define DL_MIFARE_DESFIRE_EV1_2K            0x28
#define DL_MIFARE_DESFIRE_EV1_4K            0x29
#define DL_MIFARE_DESFIRE_EV1_8K         0x2A
```

If the card type is not supported, function return the value zero.

```
#define TAG_UNKNOWN                         0x00
```

Function declaration (C language):

```
unsigned long GetDlogicCardType(unsigned char *lpucCardType);
```

Parameter description:

- lpucCardType Pointer to the variable in which the value of the card type written.

- **GetCardIdEx**

Function description:

Function writes the card serial number card in a byte array, and also gives the length of the serial number of the card. Length of the serial number can be 4 (UID size: single), 7 (UID size: double) or 10 (UID size: triple) bytes, so it is necessary to define an array of 10 bytes which will be used to enter the card serial number.

Function declaration (C language):

```
unsigned long GetCardIdEx(unsigned char *lpuSak,
          unsigned char *aucUid,
          unsigned char *lpucUidSize);
```

Parameters description:

- lpuSak Pointer to the variable in which the SAK (Select Acknowledge) written.
- AucUid Pointer to array of bytes in which the card UID (Unique Identifier) written.
- LpucUidSize Pointer to variable for the card UID size.

- **GetLastCardIdEx**
- Function description:
- Function get UID information from the uFReader about the last detected card. Function writes the card serial number card in a byte array, and also gives the length of the serial number of the card. It is necessary to define an array of 10 bytes which will be used to enter the card serial number.
- Function declaration (C language):
- ```
  unsigned long GetLastCardIdEx(unsigned char *lpuSak,
            unsigned char *aucUid,
            unsigned char *lpucUidSize);
  ```
- Parameters description:
- lpuSak Pointer to the variable in which the SAK (Select Acknowledge) written.
- AucUid Pointer to array of bytes in which the card UID (Unique Identifier) written.
- LpucUidSize Pointer to variable for the card UID size.

Function for DLL version reading

- **GetDllVersion**

Function description:

The function returns the number of type unsigned long (4 bytes). The meaning of bytes in this number are presented in the table below.

0	1	2	3
DLL Major Version	DLL Minor Version	DLL Build	

Example:

Function returns the number 0x00080202.

DLL Major Version = 2

DLL Minor Version = 2

DLL Build = 8
  ```

Function declaration (C language):

```
unsigned long GetDllVersion(void);
```

Functions that support NDEF records

- **get_ndef_record_count**

Function description:

Function returns the number of NDEF messages that have been read from the card, and number of NDEF records, number of NDEF empty messages. Also, function returns array of bytes containing number of messages pairs. First byte of pair is message ordinal, and second byte is number of NDEF records in that message. Message ordinal starts from 1.

Function declaration (C language)

```
unsigned long get_ndef_record_count(
                unsigned char *ndef_message_cnt,
                unsigned char *ndef_record_cnt,
                unsigned char *ndef_record_array,
                unsigned char *empty_ndef_message_cnt);
```

Parameters description:

- ndef_message_cnt pointer to the variable containing number of NDEF messages
- ndef_record_cnt pointer to the variable containing number of NDEF records
- ndef_record_array pointer to the array of bytes containing pairs (message ordinal – number of records)
- empty_ndef_message_cnt pointer to the variable containing number of empty messages

- **read_ndef_record**

Function description:

Function returns TNF, type of record, ID and payload from the NDEF record. NDEF record shall be elected by the message ordinal and record ordinal in this message.

Function declaration (C language)

```
unsigned long read_ndef_record(unsigned char message_nr,
                unsigned char record_nr,
                unsigned char *tnf,
                unsigned char *type_record,
                unsigned char *type_length,
                unsigned char *id,
                unsigned char *id_length,
                unsigned char *payload,
                unsigned long *payload_length);
```

Parameters description:

- message_nr — NDEF message ordinal (starts form 1)
- record_nr — NDEF record ordinal (in message)
- tnf — pointer to the variable containing TNF of record
- type_record — pointer to array containing type of record
- type_length string — pointer to the variable containing length of type of record
- id — pointer to array containing ID of record
- id_length — pointer to the variable containing length of ID of record string
- payload — pointer to array containing payload of record
- payload_length — pointer to the variable containing length of payload

- **write_ndef_record**

Function description:

Function adds a record to the end of message, if one or more records already exist in this message. If current message is empty, then this empty record will be replaced with the record. Parameters of function are: ordinal of message, TNF, type of record, ID, payload. Function also returns pointer to the variable which reported that the card

formatted for NDEF using (card does not have a capability container, for example new Mifare Ultralight, or Mifare Classic card).

Function declaration (C language)

```
unsigned long write_ndef_record(
unsigned char message_nr,
unsigned char *tnf,
unsigned char *type_record,
unsigned char *type_length,
unsigned char *id,
unsigned char *id_length,
unsigned char *payload,
unsigned long *payload_length,
unsigned char *card_formated);
```

Parameters description:

- message_nr — NDEF message ordinal (starts form 1)
- tnf — pointer to the variable containing TNF of record
- type_record — pointer to array containing type of record
- type_length string — pointer to the variable containing length of type of record
- id — pointer to array containing ID of record
- id_length — pointer to the variable containing length of ID of record string
- payload — pointer to array containing payload of record
- payload_length — pointer to the variable containing length of payload
- card_formated for NDEF using. — pointer to the variable which shows that the card formatted

- **erase_last_ndef_record**

Function description:

Function deletes the last record of selected message. If message contains one record, then it will be written empty message.

Function declaration (C language)

```
unsigned long erase_last_ndef_record(unsigned char message_nr);
```

Parameter description:

- message_nr NDEF message ordinal (starts form 1)

- **erase_all_ndef_records**

Function description:

Function deletes all records of message, then writes empty message.

Function declaration (C language)

```
unsigned long erase_all_ndef_records(unsigned char message_nr);
```

Parameter description:

- message_nr NDEF message ordinal (starts form 1)

- **ndef_card_initialization**

Function description:

Function prepares the card for NDEF using. Function writes Capability Container (CC) if necessary, and writes empty message. If card is MIFARE CLASSIC or MIFARE PLUS, then function writes MAD (MIFARE Application Directory), and default keys and access bits for NDEF using.

Function declaration (C language)

```
unsigned long ndef_card_initialization(void);
```

ERROR CODES OF NDEF FUNCTIONS

```
UFR_WRONG_NDEF_CARD_FORMAT = 0x80
UFR_NDEF_MESSAGE_NOT_FOUND = 0x81
UFR_NDEF_UNSUPPORTED_CARD_TYPE = 0x82
UFR_NDEF_CARD_FORMAT_ERROR = 0x83
UFR_MAD_NOT_ENABLED = 0x84
UFR_MAD_VERSION_NOT_SUPPORTED = 0x85
```

Functions for configuration of asynchronously card ID sending

When the card put on the reader, then the string which contains card ID shall be sent. String contains hexadecimal notation of card ID, after that is one mandatory suffix character. Before the card ID may be one prefix character placed.
When the card is removed, then shall be sent the prefix and suffix without card ID.

Example:

Card ID is 0xA103C256, prefix is 0x58 ('X'), suffix is 0x59 ('Y')

String is “XA103C256Y” when card put on reader, and “XY” when card removed.

If flag reverse byte order set, then string will be “X56C203A1Y”.

Decimal representation of card ID is enabled only for 4 bytes long card ID.

If decimal representation and reverse byte order are sets, then string will be “X1455562810Y”

- **SetAsyncCardIdSendConfig**

Function description:

Function sets the parameters of card ID sending. Parameters are: prefix existing, prefix character, suffix character, and baud rate for card ID sending.

Function declaration (C language)

```
unsigned long SetAsyncCardIdSendConfig(
unsigned char send_enable,
unsigned char prefix_enable,
unsigned char prefix,
unsigned char suffix,
unsigned char send_removed_enable,
unsigned long async_baud_rate);
```

Parameters description:

- send_enable sending enable flag (0 – disabled, 1 – enabled)
- prefix_enable prefix existing flag (0 – prefix don't exist, 1 – prefix exist)
- prefix prefix character
- suffix suffix character
- send_removed_enable flag(0 – disabled, 1 - enabled)
- async_baud_rate baud rate value (e.g. 9600)

- **GetAsyncCardIdSendConfig**

Function description:

Function returns the parameters of card ID sending.

Function declaration (C language)

```
unsigned long GetAsyncCardIdSendConfigEx(
unsigned char *send_enable,
unsigned char *prefix_enable,
unsigned char *prefix,
unsigned char *suffix,
```

```
unsigned char *send_removed_enable,
unsigned long *async_baud_rate);
```

Parameters description:

- send_enable pointer to the sending enable flag
- prefix_enable pointer to the prefix existing flag
- prefix pointer to the prefix variable
- suffix pointer to the suffix variable
- send_removed_enable pointer to the flag
- async_baud_rate pointer to the baud rate variable.

- **SetAsyncCardIdSendConfigEx**

Function description:

Function sets the parameters of card ID sending. Parameters are: prefix existing, prefix character, suffix character, and baud rate for card ID sending.

Function declaration (C language)

```
unsigned long SetAsyncCardIdSendConfig(
unsigned char send_enable,
unsigned char prefix_enable,
unsigned char prefix,
unsigned char suffix,
unsigned char send_removed_enable,
unsigned char reverse_byte_order,
unsigned char decimal_representation,
unsigned long async_baud_rate);
```

Parameters description:

- send_enable sending enable flag (0 – disabled, 1 – enabled)
- prefix_enable prefix existing flag (0 – prefix don't exist, 1 – prefix exist)
- prefix prefix character
- suffix suffix character
- send_removed_enable flag(0 – disabled, 1 – enabled)
- reverse_byte_order flag (0 – disabled, 1 – enabled)
- decimal_representation flag (0 – disabled, 1 – enabled)

- async_baud_rate baud rate value (e.g. 9600)

- **GetAsyncCardIdSendConfigEx**

Function description:

Function returns the parameters of card ID sending.

Function declaration (C language)

```
unsigned long GetAsyncCardIdSendConfigEx(
unsigned char *send_enable,
unsigned char *prefix_enable,
unsigned char *prefix,
unsigned char *suffix,
unsigned char *send_removed_enable,
unsigned char *reverse_byte_order,
unsigned char *decimal_representation,
unsigned long *async_baud_rate);
```

Parameters description:

- send_enable pointer to the sending enable flag
- prefix_enable pointer to the prefix existing flag
- prefix prefix character
- suffix suffix character
- send_removed_enable pointer to the flag
- reverse_byte_order pointer to the flag
- decimal_representation pointer to the flag
- async_baud_rate baud rate value (e.g. 9600)

Functions that works with Real Time Clock (RTC)

RTC embedded in uFR Advance device only.

- **GetReaderTime**

Function description:

Function returns 6 bytes array of unsigned char that represented current date and time into device's RTC.

- Byte 0 represent year (current year – 2000)
- Byte 1 represent month (1 – 12)
- Byte 2 represent day of the month (1 – 31)
- Byte 3 represent hour (0 – 23)
- Byte 4 represent minute (0 – 59)
- Byte 5 represent second (0 – 59)

Function declaration (C language)

```
unsigned long GetReaderTime(unsigned char *time);
```

Parameter description:

- time is pointer to the array containing current date and time representation.

- **SetReaderTime**

Function description:

Function sets the date and time into device's RTC. Function requires the 8 bytes password entry to set date and time. Date and time are represent into 6 bytes array in same way as in GetReaderTime function. Factory password is “11111111” (0x31, 0x31, 0x31, 0x31, 0x31, 0x31, 0x31, 0x31).

Function declaration (C language)

```
unsigned long SetReaderTime(
unsigned char *password,
unsigned char *time);
```

Parameters description:

- password pointer to the 8 bytes array containing password
- time pointer to the 6 bytes array containing date and time representation

- **ChangeReaderPassword**

Function description:

Function changes password for set date and time. Function's parameters are old password and new password.

Function declaration (C language)

```
unsigned long ChangeReaderPassword(
unsigned char *old_password,
unsigned char *new_password);
```

Parameters description:

- old_password pointer to the 8 bytes array containing current password
- new_password pointer to the 8 bytes array containing new password

Functions that works with EEPROM

EEPROM embedded in uFR Advance device only.

Range of user address is from 0 to 32750.

- **ReaderEepromRead**

Function description:

Function returns array of data read from EEPROM. Maximal length of array is 128 bytes.

Function declaration (C language)

```
unsigned long ReaderEepromRead(
unsigned char *data,
unsigned long address,
unsigned long size);
```

Parameters description:

- data pointer to array containing data from EEPROM
- address address of first data
- size length of array

- **ReaderEepromWrite**

Function description:

Function writes array of data into EEPROM. Maximal length of array is 128 bytes. Function requires password which length is 8 bytes. Factory password is “11111111” (0x31, 0x31, 0x31, 0x31, 0x31, 0x31, 0x31, 0x31).

Function declaration (C language)

```
unsigned long ReaderEepromWrite(
unsigned char *data,
unsigned long address,
unsigned long size,
unsigned char *password);
```

Parameters description:

- data pointer to array containing data
- address address of first data
- size length of array
- password pointer to array containing password

Functions that works with Mifare Desfire Card (AES encryption in reader)

AES encryption and decryption is performed in the reader. AES keys are stored into reader.

- **uFR_int_WriteAesKey**

Function description:

Function writes AES key (16 bytes) into reader.

Function declaration (C language)

```
unsigned long uFR_int_WriteAesKey(
unsigned char aes_key_nr,
unsigned char *aes_key);
```

Parameters description:

- aes_key_nr ordinal number of AES key in the reader
- aes_key_ext pointer to 16 byte array containing the AES key

- **uFR_int_GetDesfireUid**
- **uFR_int_GetDesfireUid_PK**

Function description:

Mifare Desfire EV1 card can be configured to use Random ID numbers instead Unique ID numbers during anti-collision procedure. In this case card uses single anti-collision loop, and returns Random Number Tag 0x08 and 3 bytes Random Number (4 bytes Random ID). This function returns Unique ID of card, if the Random ID is used.

Function declaration (C language)

```
unsigned long uFR_int_GetDesfireUid(
unsigned char aes_key_nr,
unsigned long aid,
unsigned char aid_key_nr,
unsigned char *card_uid,
unsigned char *card_uid_len,
unsigned short *card_status,
unsigned short *exec_time);
```

```
unsigned long uFR_int_GetDesfireUid_PK(
unsigned char *aes_key_ext,
unsigned long aid,
unsigned char aid_key_nr,
unsigned char *card_uid,
unsigned char *card_uid_len,
unsigned short *card_status,
unsigned short *exec_time);
```

Parameters description:

- aes_key_nr ordinal number of AES key in the reader
- aes_key_ext pointer to 16 byte array containing the AES key
- aid ID of application that uses this key (3 bytes long, 0x000000 for card master key)
- aid_key_nr key number into application (0 for card master key or application master key)
- data pointer to array containing card UID
- data_len pointer to card UID length variable
- card_status pointer to card error variable
- exec_time function's execution time

- **ufr_int_DesfireFreeMem**

Function description:

Function returns the available bytes on the card.

Function declaration (C language)

```
unsigned long uFR_int_DesfireFreeMem(
unsigned long *free_mem_byte,
unsigned short *card_status,
unsigned short *exec_time);
```

Parameters description:

- free_mem_byte pointer to free memory size variable
- card_status pointer to card error variable
- exec_time function's execution time

- **uFR_int_DesfireFormatCard**
- **uFR_int_DesfireFormatCard_PK**

Function description:

Function releases all allocated user memory on the card. All applications will be deleted, also all files within those applications will be deleted. Only the card master key, and card master key settings will not be deleted. This operation requires authentication with the card master key.

Function declaration (C language)

```
unsigned long uFR_int_DesfireFormatCard(
unsigned char aes_key_nr,
unsigned short *card_status,
unsigned short *exec_time);
```

```
unsigned long uFR_int_DesfireFormatCard_PK(
unsigned char *aes_key_ext,
unsigned short *card_status,
unsigned short *exec_time);
```

Parameters description:

- aes_key_nr ordinal number of card master AES key in the reader

- aes_key_ext pointer to 16 byte array containing the AES key
- card_status pointer to card error variable
- exec_time function's execution time

- **uFR_int_DesfireSetConfiguration**
- **uFR_int_DesfireSetConfiguration_PK**

Function description:

Function allows you to activate the Random ID option, and/or Format disable option. If these options are activated, then they can not be returned to the factory setting (Random ID disabled, Format card enabled). This operation requires authentication with the card master key.

Function declaration (C language)

```
unsigned long uFR_int_DesfireSetConfiguration(
unsigned char aes_key_nr,
unsigned char random_uid,
unsigned char format_disable,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireSetConfiguration_PK(
unsigned char *aes_key_ext,
unsigned char random_uid,
unsigned char format_disable,
unsigned short *card_status,
unsigned short *exec_time);
```

Parameters description:

- aes_key_nr ordinal number of card master AES key in the reader
- aes_key_ext pointer to 16 byte array containing the AES key
- random_uid 0 – Random ID disabled, 1 – Random ID enabled
- format_disable 0 – Format enabled, 1 – Format disabled
- card_status pointer to card error variable
- exec_time function's execution time

- **uFR_int_DesfireGetKeySettings**
- **uFR_int_DesfireGetKeySettings_PK**

Function description:

Function allows to get card master key and application master key configuration settings. In addition it returns the maximum number of keys which can be stored within selected application.

Function declaration (C language)

```
unsigned long uFR_int_DesfireGetKeySettings(
unsigned char aes_key_nr,
unsigned long aid,
unsigned char *settings
unsigned char *max_key_no,
unsigned short *card_status,
unsigned short *exec_time);
```

```
unsigned long uFR_int_DesfireGetKeySettings_PK(
unsigned char *aes_key_ext,
unsigned long aid,
unsigned char *settings
unsigned char *max_key_no,
unsigned short *card_status,
unsigned short *exec_time);
```

Parameters description:

- aes_key_nr ordinal number of AES key in the reader
- aes_key_ext pointer to 16 byte array containing the AES key
- aid ID of application that uses this key (3 bytes long, 0x000000 for card master key)
- settings pointer to settings variable
- max_key_no maximum number of keys within selected application
- card_status pointer to card error variable
- exec_time function's execution time

- **uFR_int_DesfireChangeKeySettings**
- **uFR_int_DesfireChangeKeySettings_PK**

Function description:

Function allows to set card master key, and application master key configuration settings.

Function declaration (C language)

```
unsigned long uFR_int_DesfireChangeKeySettings(
unsigned char aes_key_nr,
unsigned long aid,
unsigned char settings,
unsigned short *card_status,
unsigned short *exec_time);
```

```
unsigned long uFR_int_DesfireChangeKeySettings_PK(
unsigned char *aes_key_ext,
unsigned long aid,
unsigned char settings,
unsigned short *card_status,
unsigned short *exec_time);
```

Parameters description:

- aes_key_nr ordinal number of AES key in the reader
- aes_key_ext pointer to 16 byte array containing the AES key
- aid ID of application that uses this key (3 bytes long, 0x000000 for card master key)
- settings pointer to key settings variable
- card_status pointer to card error variable
- exec_time function's execution time

- **uFR_int_DesfireChangeAesKey**
- **uFR_int_DesfireChangeAesKey_PK**

Function description:

Function allow to change any AES key on the card. Changing the card master key require current card master key authentication. Authentication for the application keys changing depend on the application master key settings (which key uses for authentication).

Function declaration (C language)

```
unsigned long uFR_int_DesfireChangeAesKey(
unsigned char aes_key_nr,
unsigned long aid,
unsigned char aid_key_nr_auth,
unsigned char new_aes_key[16],
unsigned char aid_key_no,
unsigned char old_aes_key[16],
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireChangeAesKey_PK(
unsigned char *aes_key_ext,
unsigned long aid,
unsigned char aid_key_nr_auth,
unsigned char new_aes_key[16],
unsigned char aid_key_no,
unsigned char old_aes_key[16],
unsigned short *card_status,
unsigned short *exec_time);
```

Parameters description:

- aes_key_nr ordinal number of AES key in the reader
- aes_key_ext pointer to 16 byte array containing the AES key
- aid ID of application that uses this key (3 bytes long, 0x000000 for card master key)
- aid_key_nr_auth key number into application which uses for authentication
- new_aes_key[16] 16 bytes array that represent AES key
- aid_key_no key number into application that will be changed
- old_aes_key[16] 16 bytes array that represent current AES key that will be changed, if this is not key by which is made authentication.
- card_status pointer to card error variable
- exec_time function's execution time

- **uFR_int_DesfireCreateAesApplication**
- **uFR_int_DesfireCreateAesApplication_PK**
- **uFR_int_DesfireCreateAesApplication_no_auth**

Function description:

Function allows to create new application on the card. Is the card master key authentication is required, depend on the card master key settings. Maximal number of applications on the card is 28. Each application is linked to set of up 14 different user definable access keys.

Function declaration (C language)

```
unsigned long uFR_int_DesfireCreateApplication(
unsigned char aes_key_nr,
unsigned long aid_nr,
unsigned char settings,
unsigned char max_key_no,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireCreateApplication_PK(
unsigned char *aes_key_ext,
unsigned long aid_nr,
unsigned char settings,
unsigned char max_key_no,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireCreateApplication_no_auth(
unsigned long aid_nr,
unsigned char settings,
unsigned char max_key_no,
unsigned short *card_status,
unsigned short *exec_time);
```

Parameters description:

- aes_key_nr — ordinal number of card master AES key in the reader
- aes_key_ext — pointer to 16 byte array containing the AES key
- aid_nr — ID of application that creates (3 bytes long 0x000000 to 0xFFFFFF)
- settings — application master key settings
- max_key_no — maximal number of keys into application
- card_status — pointer to card error variable
- exec_time — function's execution time

- **uFR_int_DesfireDeleteApplication**
- **uFR_int_DesfireDeleteApplication_PK**

Function description:

Function allows to deactivate application on the card. Is the card master key authentication is required, depend on the card master key settings. AID allocation is removed, but deleted memory blocks can only recovered by using Format card function.

Function declaration (C language)

```
unsigned long uFR_int_DesfireDeleteApplication(
unsigned char aes_key_nr,
unsigned long aid_nr,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireDeleteApplication_PK(
unsigned char *aes_key_ext,
unsigned long aid_nr,
unsigned short *card_status,
unsigned short *exec_time);
```

Parameters description:

- aes_key_nr — ordinal number of card master AES key in the reader
- aes_key_ext — pointer to 16 byte array containing the AES key
- aid_nr — ID of application that deletes (3 bytes long 0x000000 to 0xFFFFFF)
- card_status — pointer to card error variable
- exec_time — function's execution time

- **uFR_int_DesfireCreateStdDataFile**
- **uFR_int_DesfireCreateStdDataFile_PK**
- **uFR_int_DesfireCreateStdDataFile_no_auth**

Function description:

Function allows to create file for the storage unformatted user data within existing application on the card. Maximal number of files into application is 32. The file will be created in the currently selected application. Is the application master key authentication

is required, depend on the application master key settings.
Communication settings define communication mode between reader and card. The communication modes are:
- plain communication communication settings value is 0x00
- plain communication secured by MACing communication settings value is 0x01
- fully enciphered communication communication settings value is 0x11
Access rights for read, write, read&write and changing, references certain key within application's keys (0 – 13). If value is 14, this means free access, independent of previous authentication. If value is 15, this means deny access (for example if write access is 15 then the file type is read only).

Function declaration (C language)

```
unsigned long uFR_int_DesfireCreateStdDataFile(
unsigned char aes_key_nr,
unsigned long aid,
unsigned char file_id,
unsigned long file_size,
unsigned char read_key_no,
unsigned char write_key_no,
unsigned char read_write_key_no,
unsigned char change_key_no,
unsigned char communication_settings,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireCreateStdDataFile_PK(
unsigned char *aes_key_ext,
unsigned long aid,
unsigned char file_id,
unsigned long file_size,
unsigned char read_key_no,
unsigned char write_key_no,
unsigned char read_write_key_no,
unsigned char change_key_no,
unsigned char communication_settings,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireCreateStdDataFile_no_auth(
unsigned long aid,
unsigned char file_id,
```

```
unsigned long file_size,
unsigned char read_key_no,
unsigned char write_key_no,
unsigned char read_write_key_no,
unsigned char change_key_no,
unsigned char communication_settings,
unsigned short *card_status,
unsigned short *exec_time);
```

Parameters description:

- aes_key_nr ordinal number of AES key in the reader
- aes_key_ext pointer to 16 byte array containing the AES key
- aid ID of application that contains the file
- file_id ID of file that will be created (0 – 31)
- file_size file size in bytes
- read_key_no key for reading
- write_key_no key for writing
- read&write_key_no key for reading and writing
- change_key_no key for changing this setting
- communication_settings variable that contains communication settings
- card_status pointer to card error variable
- exec_time function's execution time

- **uFR_int_DesfireDeleteFile**
- **uFR_int_DesfireDeleteFile_PK**
- **uFR_int_DesfireDeleteFile_no_auth**

Function description:

Function deactivates a file within currently selected application. Allocated memory blocks associated with deleted file not set free. Only format card function can delete the memory blocks. Is the application master key authentication is required, depend on the application master key settings.

Function declaration (C language)

```
unsigned long uFR_int_DesfireDeleteFile(
unsigned char aes_key_nr,
unsigned long aid,
unsigned char file_id,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireDeleteFile_PK(
unsigned char *aes_key_ext,
unsigned long aid,
unsigned char file_id,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireDeleteFile_no_auth(
unsigned long aid,
unsigned char file_id,
unsigned short *card_status,
unsigned short *exec_time);
```

Parameters description:

- internal_key 1 - AES key stored into reader uses for authentication, 0 – AES key uses for authentication entered as a function's parameter
- aes_key_nr ordinal number of AES key in the reader
- aes_key_ext pointer to 16 byte array containing the AES key
- aid ID of application that contains the file
- file_id ID of file that will be deleted (0 – 31)
- auth_req value is 1 if authentication is needed, otherwise it is 0
- card_status pointer to card error variable
- exec_time function's execution time

- **uFR_int_DesfireReadStdDataFile**
- **uFR_int_DesfireReadStdDataFile_PK**
- **uFR_int_DesfireReadStdDataFile_no_auth**

Function description:

Function allow to read data from Standard Data File, or from Backup Data File. Read command requires a preceding authentication either with the key specified for Read or Read&Write access.

Function declaration (C language)

```
unsigned long uFR_int_DesfireReadStdDataFile(
unsigned char aes_key_nr,
unsigned long aid,
unsigned char aid_key_nr,
unsigned char file_id,
unsigned short offset,
unsigned short data_length,
unsigned char communication_settings,
unsigned char *data,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireReadStdDataFile_PK(
unsigned char *aes_key_ext,
unsigned long aid,
unsigned char aid_key_nr,
unsigned char file_id,
unsigned short offset,
unsigned short data_length,
unsigned char communication_settings,
unsigned char *data,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireReadStdDataFile_no_auth(
unsigned long aid,
unsigned char aid_key_nr,
unsigned char file_id,
unsigned short offset,
unsigned short data_length,
unsigned char communication_settings,
unsigned char *data,
unsigned short *card_status,
unsigned short *exec_time);
```

Parameters description:

- aes_key_nr ordinal number of AES key in the reader

- aes_key_ext pointer to 16 byte array containing the AES key
- aid ID of application that contains the file
- aid_key_nr key number into application
- file_id ID of file (0 – 31)
- offset start position for read operation within file
- data_length number of data to be read
- communication_settings value must be same as in file declaration
- data pointer to data array
- card_status pointer to card error variable
- exec_time function's execution time

- **uFR_int_DesfireWriteStdDataFile**
- **uFR_int_DesfireWriteStdDataFile_PK**
- **uFR_int_DesfireWriteStdDataFile_no_auth**

Function description:

Function allow to write data to Standard Data File, or to Backup Data File. Write command requires a preceding authentication either with the key specified for Write or Read&Write access.

Function declaration (C language)

```
unsigned long uFR_int_DesfireWriteStdDataFile(
unsigned char aes_key_nr,
unsigned long aid,
unsigned char aid_key_nr,
unsigned char file_id,
unsigned short offset,
unsigned short data_length,
unsigned char communication_settings,
unsigned char *data,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireWriteStdDataFile_PK(
unsigned char *aes_key_ext,
unsigned long aid,
unsigned char aid_key_nr,
```

```
unsigned char file_id,
unsigned short offset,
unsigned short data_length,
unsigned char communication_settings,
unsigned char *data,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireWriteStdDataFile_no_auth(
unsigned long aid,
unsigned char aid_key_nr,
unsigned char file_id,
unsigned short offset,
unsigned short data_length,
unsigned char communication_settings,
unsigned char *data,
unsigned short *card_status,
unsigned short *exec_time);
```

Parameters description:

- aes_key_nr ordinal number of AES key in the reader
- aes_key_ext pointer to 16 byte array containing the AES key
- aid ID of application that contains the file
- aid_key_nr key number into application
- file_id ID of file (0 – 31)
- offset start position for write operation within file
- data_length number of data to be written
- communication_settings value must be same as in file declaration
- data pointer to data array
- card_status pointer to card error variable
- exec_time function's execution time

- **DES_to_AES_key_type**

Function description:

Function allow to change the card master key type from DES to AES. Factory setting for DESFIRE card master key is DES key type, and value is 0x0000000000000000.

Because the reader uses AES keys, you must change the type key on AES. New AES key is 0x00000000000000000000000000000000.

Function declaration (C language)

```
unsigned long DES_to_AES_key_type(void);
```

- **uFR_int_DesfireCreateValueFile**
- **uFR_int_DesfireCreateValueFile_PK**
- **uFR_int_DesfireCreateValueFile_no_auth**

Function description:

Function allows to create file for the storage and manipulation of 32bit signed integer values within existing application on the card. Maximal number of files into application is 32. The file will be created in the currently selected application. Is the application master key authentication is required, depend on the application master key settings. Communication settings define communication mode between reader and card. The communication modes are:
- plain communication communication settings value is 0x00
- plain communication secured by MACing communication settings value is 0x01
- fully enciphered communication communication settings value is 0x11
Access rights for read, write, read&write and changing, references certain key within application's keys (0 – 13). If value is 14, this means free access, independent of previous authentication. If value is 15, this means deny access (for example if write access is 15 then the file type is read only).

Function declaration (C language)

```
unsigned long uFR_int_DesfireCreateValueFile(
unsigned char aes_key_nr,
unsigned long aid,
unsigned char file_id,
long lower_limit,
long upper_limit,
long value,
unsigned char limited_credit_enabled,
unsigned char read_key_no,
unsigned char write_key_no,
unsigned char read_write_key_no,
```

```
unsigned char change_key_no,
unsigned char communication_settings,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireCreateValueFile_PK(
unsigned char *aes_key_ext,
unsigned long aid,
unsigned char file_id,
long lower_limit,
long upper_limit,
long value,
unsigned char limited_credit_enabled,
unsigned char read_key_no,
unsigned char write_key_no,
unsigned char read_write_key_no,
unsigned char change_key_no,
unsigned char communication_settings,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireCreateValueFile_no_auth(
unsigned long aid,
unsigned char file_id,
long lower_limit,
long upper_limit,
long value,
unsigned char limited_credit_enabled,
unsigned char read_key_no,
unsigned char write_key_no,
unsigned char read_write_key_no,
unsigned char change_key_no,
unsigned char communication_settings,
unsigned short *card_status,
unsigned short *exec_time);
```

Parameters description:

- aes_key_nr ordinal number of AES key in the reader
- aes_key_ext pointer to 16 byte array containing the AES key
- aid ID of application that contains the file
- file_id ID of file that will be created (0 – 31)

- lower_limit lower limit which is valid for this file
- upper_limit upper limit which is valid for this file
- value initial value of the value file
- limited_credit_enabled bit 0 – limited credit enabled (1 – yes, 0 – no)
  bit 1 – free get value (1 – yes, 0 – no)
- read_key_no key for get and debit value
- write_key_no key for get, debit and limited credit value
- read&write_key_no key for get, debit, limited credit and credit value
- change_key_no key for changing this setting
- communication_settings variable that contains communication settings
- card_status pointer to card error variable
- exec_time function's execution time

- **uFR_int_DesfireReadValueFile**
- **uFR_int_DesfireReadValueFile_PK**
- **uFR_int_DesfireReadValueFile_no_auth**

Function description:

Function allows to increase a value stored in a value files. Read command requires a preceding authentication either with the key specified for Read or Read&Write access.

Function declaration (C language)

```
unsigned long uFR_int_DesfireReadValueFile(
unsigned char aes_key_nr,
unsigned long aid,
unsigned char aid_key_nr,
unsigned char communication_settings,
long *value,
unsigned short *card_status,
unsigned short *exec_time);
```

```
unsigned long uFR_int_DesfireReadValueFile_PK(
unsigned char *aes_key_ext,
unsigned long aid,
unsigned char aid_key_nr,
unsigned char communication_settings,
long *value,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireReadValueFile_no_auth(
unsigned long aid,
unsigned char aid_key_nr,
unsigned char communication_settings,
long *value,
unsigned short *card_status,
unsigned short *exec_time);
```

Parameters description:

- aes_key_nr ordinal number of AES key in the reader
- aes_key_ext pointer to 16 byte array containing the AES key
- aid ID of application that contains the file
- aid_key_nr key number into application
- file_id ID of file (0 – 31)
- communication_settings value must be same as in file declaration
- value pointer to value variable
- card_status pointer to card error variable
- exec_time function's execution time

- **uFR_int_DesfireIncreaseValueFile**
- **uFR_int_DesfireIncreaseValueFile_PK**
- **uFR_int_DesfireIncreaseValueFile_no_auth**

Function description:

Function allows to decrease a value stored in a value files. Read command requires a preceding authentication either with the key specified for Read or Read&Write access.

Function declaration (C language)

```
unsigned long uFR_int_DesfireIncreaseValueFile(
unsigned char aes_key_nr,
unsigned long aid,
unsigned char aid_key_nr,
unsigned char communication_settings,
long value,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireIncreaseValueFile_PK(
unsigned char *aes_key_ext,
unsigned long aid,
unsigned char aid_key_nr,
unsigned char communication_settings,
long value,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireIncreaseValueFile_no_auth(
unsigned long aid,
unsigned char aid_key_nr,
unsigned char communication_settings,
long *value,
unsigned short *card_status,
unsigned short *exec_time);
```

Parameters description:

- aes_key_nr ordinal number of AES key in the reader
- aes_key_ext pointer to 16 byte array containing the AES key
- aid ID of application that contains the file
- aid_key_nr key number into application
- file_id ID of file (0 – 31)
- communication_settings value must be same as in file declaration
- value value (must be positive number)
- card_status pointer to card error variable
- exec_time function's execution time

- **uFR_int_DesfireDecreaseValueFile**

- **uFR_int_DesfireDecreaseValueFile_PK**
- **uFR_int_DesfireDecreaseValueFile_no_auth**

Function description:

Function allow to read value from value files. Read command requires a preceding authentication either with the key specified for Read or Read&Write access.

Function declaration (C language)

```
unsigned long uFR_int_DesfireDecreaseValueFile(
unsigned char aes_key_nr,
unsigned long aid,
unsigned char aid_key_nr,
unsigned char communication_settings,
long value,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireDecreaseValueFile_PK(
unsigned char *aes_key_ext,
unsigned long aid,
unsigned char aid_key_nr,
unsigned char communication_settings,
long value,
unsigned short *card_status,
unsigned short *exec_time);

unsigned long uFR_int_DesfireDecreaseValueFile_no_auth(
unsigned long aid,
unsigned char aid_key_nr,
unsigned char communication_settings,
long *value,
unsigned short *card_status,
unsigned short *exec_time);
```

Parameters description:

- aes_key_nr — ordinal number of AES key in the reader
- aes_key_ext — pointer to 16 byte array containing the AES key
- aid — ID of application that contains the file
- aid_key_nr — key number into application
- file_id — ID of file (0 – 31)

- communication_settings value must be same as in file declaration
- value value (must be positive number)
- card_status pointer to card error variable
- exec_time function's execution time

Support for ISO14443-4A protocol

The protocol defines three fundamental types of blocks:

- I-block used to convey information for use by the application layer.

- R-block used to convey positive or negative acknowledgements. An R-block never contains an INF field. The acknowledgement relates to the last received block.

- S-block used to exchange control information between the PCD and the PICC. Two different types of S-blocks

are defined:

1) Waiting time extension containing a 1 byte long INF field and

2) DESELECT containing no INF field.

Function declarations:

```
UFR_STATUS DL_API i_block_trans_rcv_chain(uint8_t chaining,
           uint8_t timeout, uint8_t block_length,
           uint8_t *snd_data_array, uint8_t *rcv_length,
           uint8_t *rcv_data_array, uint8_t *rcv_chained,
           uint32_t *ufr_status);
```

Parameters description:

- chaining – 1- chaining uses, 0 – no chaining
- timeout - timeout for card reply
- block_length - inf block length
- snd_data_array - pointer to array of data that will be send
- rcv_length - length of received data
- rcv_data_array - pointer to array of data that will be received
- rcv_chained - 1 received packet is chained, 0 received packet is not chained
- urf_status – card operation status

```
UFR_STATUS DL_API r_block_transceive(uint8_t ack, uint8_t timeout,
           uint8_t *rcv_length, uint8_t *rcv_data_array,
           uint8_t *rcv_chained, uint32_t *ufr_status);
```

Parameter description:

- ack - 1 ACK, 0 NOT ACK
- timeout - timeout for card reply
- rcv_length - length of received data
- rcv_data_array - pointer to array of data that will be received
- rcv_chained - 1 received packet is chained, 0 received packet is not chained
- urf_status – card operation status

```
UFR_STATUS DL_API s_block_deselect(uint8_t timeout);
```

- timeout – timeout in [ms]

Support for APDU commands in ISO 14443-4A tags

Some ISO 14443-4A tags supports the APDU message structure according to ISO/IEC 7816-4.

For more details you have to check the manual for the tags that you planing to use.

Function declarations used to support APDU message structure:

```
UFR_STATUS SetISO14443_4_Mode(void);
UFR_STATUS uFR_APDU_Transceive(uint8_t cls, uint8_t ins, uint8_t p0, uint8_t p1,
              uint8_t *data_out, uint8_t data_out_len, uint8_t *data_in, uint32_t
              max_data_in_len, uint32_t *response_len,
              uint8_t send_le, uint8_t *apdu_status);
UFR_STATUS s_block_deselect(uint8_t timeout);
```

Parameters description:

- cls – APDU CLA (class byte)
- ins – APDU command code (instruction byte)
- p1, p2 - parameter bytes
- data_out - APDU command data field. Use NULL if data_out_len is 0
- data_out_len – number of bytes in the APDU command data field (Lc field)
- *data_in - buffer for receiving APDU response. There should be allocated at least (send_le + 2) bytes before function call.

- max_data_in_len size of receiving buffer. If the APDU response exceeded size of buffer, then function returns error
- *response_len – value of the Le fied if send_le is not 0. After successful execution location pointed by response_len will contain number of bytes in the APDU response.
- send_le – if this parameter is 0 then APDU Le field will not be sent. Otherwise Le field will be included in the APDU message. Value response_len pointed to before function call will be value of the Le field.
- apdu_status - APDU error codes SW1 and SW2 in 2 bytes array.

To send APDU message you have to follow next procedure:

1. Call SetISO14443_4_Mode(). ISO 14443-4A tag in a field will be selected and RF field polling will be stopped.
2. Call uFR_APDU_Transceive() as many times as you needed.
3. Call s_block_deselect() to deselect tag and restore RF field polling. This call is mandatory.

Fully uFR firmware support for APDU commands in ISO 14443-4A tags

This group of newly designed functions makes use of the uFR_APDU_Transceive() obsolete. However, uFR_APDU_Transceive() function is still part of the uFCoder library for backward compatibility.

New functions implemented in the uFCoder library are:

```
UFR_STATUS APDUHexStrTransceive(const char *c_apdu,
                                char **r_apdu);

UFR_STATUS APDUPlainTransceive(const uint8_t *c_apdu,
                               uint32_t c_apdu_len,
                               uint8_t *r_apdu,
                               uint32_t *r_apdu_len);

UFR_STATUS APDUTransceive(uint8_t cls,
                          uint8_t ins,
                          uint8_t p0,
                          uint8_t p1,
                          const uint8_t *data_out,
                          uint32_t Nc,
                          uint8_t *data_in,
                          uint32_t *Ne,
                          uint8_t send_le,
                          uint8_t *apdu_status);
```

These functions are more responsive than obsolete uFR_APDU_Transceive(), because most of the work is performed by a uFR firmware.

- **`UFR_STATUS APDUHexStrTransceive(const char *c_apdu,`**
  `char **r_apdu);`

Using this function, you can send C–APDU in the c_string (zero terminated) containing pairs of the hexadecimal digits. Pairs of the hexadecimal digits can be delimited by any of the punctuation characters or white space.

**r_apdu returns pointer to the c_string (zero terminated) containing pairs of the hexadecimal digits without delimiters.

- **`UFR_STATUS APDUPlainTransceive(const uint8_t *c_apdu,`**
  `uint32_t c_apdu_len,`
  `uint8_t *r_apdu,`
  `uint32_t *r_apdu_len);`

This is binary alternative function to the **`APDUHexStrTransceive()`**. C-APDU and R-APDU are sent and receive in the form of the byte arrays. There is obvious need for a **`c_apdu_len`** and **`*r_apdu_len`** parameters which represents length of the **`*c_apdu`** and **`*r_apdu`** byte arrays, respectively.

The memory space on which **`*r_apdu`** points, have to be allocated before calling of the **`APDUPlainTransceive()`**. Number of the bytes allocated have to correspond to the N_e bytes, defined by the L_e field in the C–APDU plus 2 bytes for SW1 and SW2.

- **`UFR_STATUS APDUTransceive(uint8_t cls,`**
  `uint8_t ins,`
  `uint8_t p0,`
  `uint8_t p1,`
  `const uint8_t *data_out,`
  `uint32_t Nc,`
  `uint8_t *data_in,`
  `uint32_t *Ne,`
  `uint8_t send_le,`
  `uint8_t *apdu_status);`

This is “exploded binary” alternative function intended for support APDU commands in ISO 14443-4A tags. **`APDUTransceive()`** receives separated parameters which are an integral part of the C–APDU. There is parameters **`cls`**, **`ins`**, **`p0`**, **`p1`** of the **`uint8_t`** type.

N_c defines number of bytes in the byte array **`*data_out`** point to. N_c also defines L_c field in the C–APDU (If $N_c == 256$ *then* $L_c = 0$, *otherwise* $L_c = N_c$). If $N_c = 0$ *then* L_c *is omitted and* ***`*data_out`*** can be NULL.

`send_le` and ***N_e*** parameters defines L_c field in the C–APDU. If **`send_le`** is 1 then L_c field will be included in the C–APDU. If **`send_le`** is 0 then L_c field will be omitted from the C–APDU.

If $*N_e == 256$ *then* $L_e = 0$, *otherwise* $L_e = *N_e$.

The memory space on which **`*data_in`**, have to be allocated before calling of the **`APDUPlainTransceive()`**. Number of the bytes allocated have to correspond to the $*N_e$ bytes, defined by the L_e field in the C–APDU.

After successfully executed **`APDUTransceive()`**, **`*data_in`** will contain R-APDU data field (body).

`*apdu_status` will contain R-APDU trailer (SW1 and SW2 APDU status bytes).

Functions for for the operating parameters of the reader setting

- **UfrSetBadSelectCardNrMax**

Function description:

The function allows you to set the number of unsuccessful card selections before it can be considered that the card is not placed on the reader. Period between two card selections is approximately 10ms. Default value of this parameter is 20 i.e. 200ms. This parameter can be set in the range of 0 to 254.
This is useful for asynchronous card ID transmission, if parameter send_removed_enable in function SetAsyncCardIdSendConfig is set. Then you can set a lower value of the number of unsuccessful card selections, in order to send information to the card removed was faster.
A small value of this parameter may cause a false report that the card is not present, and immediately thereafter true report that the card is present.

Function declaration (C language)

```
unsigned long UfrSetBadSelectCardNrMax(
unsigned char bad_select_nr_max);
```

- **UfrGetBadSelectCardNrMax**

Function description:

The function returns value of maximal unsuccessful card selections, which is set in reader.

Function declaration (C language)

```
unsigned long UfrGetBadSelectCardNrMax(
unsigned char *bad_select_nr_max);
```

Functions for all blocks linear reading

Functions allow you to quickly read data from the card including the sector trailer blocks. These functions are very similar to the functions for linear reading of users data space.

- ***LinearRowRead***
- ***LinearRowRead_AKM1***
- ***LinearRowRead_AKM2***
- ***LinearRowRead_PK***

Functions declaration (C language):

```
unsigned long LinearRowRead(unsigned char *aucData,
                  unsigned short usLinearAddress,
                  unsigned short usDataLength,
                  unsigned short *lpusBytesReturned,
                  unsigned char ucAuthMode,
                  unsigned char ucReaderKeyIndex);

unsigned long LinearRowRead_AKM1(unsigned char *aucData,
                           unsigned short usLinearAddress,
                           unsigned short usDataLength,
                           unsigned short *lpusBytesReturned,
                           unsigned char ucAuthMode);

unsigned long LinearRowRead_AKM2(unsigned char *aucData,
                          unsigned short usLinearAddress,
                          unsigned short usDataLength,
                          unsigned short *lpusBytesReturned,
                          unsigned char ucAuthMode);

unsigned long LinearRowRead_PK(unsigned char *aucData,
                           unsigned short usLinearAddress,
                           unsigned short usDataLength,
                           unsigned short *lpusBytesReturned,
                           unsigned char ucAuthMode,
                           unsigned char *aucProvidedKey);
```

The method for proving authenticity is determined by the suffix in the functions names:

- *aucData* - Pointer to the sequence of bytes where read data will be stored.
- *usLinearAddress* - Linear address on the card from which the data want to read
- *usDataLength* - Number of bytes for reading. For aucData a minimum usDataLength bytes must be allocated before calling the function

- *lpusBytesReturned* - Pointer to "unsigned short" type variable, where the number of successfully read bytes from the card is written. If the reading is fully managed this data is equal to the usDataLength parameter. If there is an error reading some of the blocks, the function returns all successfully read data in the aucData before the errors occurrence and the number of successfully read bytes is returned via this parameter

- *ucAuthMode* – This parameter ***for Mifare Classic*** *tags defines whether to perform authentication with key A or key B. It can have two values, namely: AUTHENT1A (0x60) or AUTHENT1B (0x61).* ***For NTAG 21x, Ultralight EV1 and other T2T tags supporting PWD_AUTH*** *value 0x61 means "****use PWD_AUTH****" with LinearRowRead() or LinearRowRead_PK() functions. Value 0x60 with LinearRowRead() or LinearRowRead_PK() functions means "****without PWD_AUTH****" and in that case you can send for ucReaderKeyIndex or aucProvidedKey parameters anything you want without influence on the result. For NTAG 21x, Ultralight EV1 and other T2T tags supporting PWD_AUTH you can use _AKM1 or _AKM2 function variants only* ***without PWD_AUTH*** *in any case of the valid values (0x60 or 0x61) provided for this parameter.*

  - ucReaderKeyIndex - The default method of authentication (when the functions without a suffix is used) performs the authenticity proving by using the selected key index from the reader. In the linear address mode, this applies to all sectors that are read

  - *aucProvidedKey* - Pointer to the six-byte string containing the key for authenticity proving in the "Provided Key" method. _PK Suffix in the name of the function indicates this method usage.

FUNCTIONS FOR READER LOW POWER MODE CONTROL

- **UfrEnterSleepMode**

Function allows enter to reader low power working mode. Reader is in sleep mode. RF field is turned off. The reader is waiting for the command to return to normal working mode.

Functions declaration (C language):

```
unsigned long UfrEnterSleepMode(void);
```

- UfrLeaveSleepMode

Function allows return from low power reader mode to normal working mode.

Functions declaration (C language):

```
unsigned long UfrLeaveSleepMode(void);
```

AutoSleepSet

from firmware version 3.8.18 and lib version 3.8.15

This function permanently set auto-sleep functionality of the device. Valid **`seconds_wait`** range is from 1 to 254. To permanently disable auto-sleep functionality use **0** or **0xFF** for the **`seconds_wait`** parameter.

Function declaration (C language):

```
unsigned long AutoSleepSet(uint8_t seconds_wait);
```

AutoSleepGet

from firmware version 3.8.18 and lib version 3.8.15

This function uses to get auto-sleep functionality setup from the device. You have to send pointer to already allocated variable of the **`uint8_t`** type. If auto-sleep functionality is disabled you will get **0** or **0xFF** in the variable pointed by the **`*seconds_wait`** parameter.

Function declaration (C language):

```
unsigned long AutoSleepGet(uint8_t *seconds_wait);
```

Functions for Reader NTAG Emulation Mode

WriteEmulationNdef

Function description:

Function store a message record for NTAG emulation mode in to the reader. Parameters of the function are: TNF, type of record, ID, payload.

Function declaration (C language):

```
unsigned long WriteEmulationNdef(uint8_t tnf,
        uint8_t* type_record,
        uint8_t type_length,
        uint8_t* id,
        uint8_t id_length,
        uint8_t* payload,
        uint8_t payload_length);
```

Parameters description:

- tnf — TNF of the record
- type_record — pointer to the array containing record type
- type_length — length of the record type
- id — pointer to the array containing record ID
- id_length — length of the record ID
- payload — pointer to the array containing record payload
- payload_length — length of the record payload

Possible error codes:

```
WRITE_VERIFICATION_ERROR = 0x70
MAX_SIZE_EXCEEDED = 0x10
```

TagEmulationStart

Put the reader permanently in a NDEF tag emulation mode. Only way for a reader to exit from this mode is to receive the TAG_EMULATION_STOP command (issued by calling **`TagEmulationStop()`** function).

In this mode, the reader can only answer to the commands issued by a following library functions:

`TagEmulationStart()`,

`WriteEmulationNdef()`,

TagEmulationStop(),

GetReaderSerialNumber(),

GetReaderSerialDescription(),

GetReaderHardwareVersion(),

GetReaderFirmwareVersion(),

GetBuildNumber()

Calls to the other functions in this mode returns following error code:

FORBIDDEN_IN_TAG_EMULATION_MODE = 0x90

Function declaration (C language):

```
unsigned long TagEmulationStart(void);
```

Possible error codes:

WRITE_VERIFICATION_ERROR = 0x70

(command resulting in a direct write to a device non-volatile memory)

TagEmulationStop

Allows the reader permanent exit from a NDEF tag emulation mode.

Function declaration (C language):

```
unsigned long TagEmulationStop(void);
```

Possible error codes:

WRITE_VERIFICATION_ERROR = 0x70

(command resulting in a direct write to a device non-volatile memory)

Functions supporting Ad-Hoc emulation mode

This mode enables user controlled emulation from the user application. There is “nfc-rfid-reader-sdk/ufr-examples-ad_hoc_emulation-c” console example written in C, which demonstrate usage of this functions.

AdHocEmulationStart

Put uFR in emulation mode with ad-hoc emulation parameters (see. SetAdHocEmulationParams() and GetAdHocEmulationParams() functions). uFR stays in ad-hoc emulation mode until AdHocEmulationStop() is called or reader reset.

Function declaration (C language):

```
UFR_STATUS AdHocEmulationStart(void);
```

AdHocEmulationStop

Terminate uFR ad-hoc emulation mode.

Function declaration (C language):

```
UFR_STATUS AdHocEmulationStop(void);
```

GetExternalFieldState

Returns external field state when uFR is in ad-hoc emulation mode.

Function declaration (C language):

```
UFR_STATUS GetExternalFieldState(uint8_t *is_field_present);
```

- is_field_present contains 0 if external field isn’t present or 1 if field is present.

GetAdHocEmulationParams

This function returns current ad-hoc emulation parameters. On uFR power on or reset ad-hoc emulation parameters are set back to their default values.

Function declaration (C language):

```
UFR_STATUS GetAdHocEmulationParams(uint8_t *ThresholdMinLevel, uint8_t
*ThresholdCollLevel, uint8_t *RFLevelAmp, uint8_t *RxGain, uint8_t
*RFLevel);
```

- ThresholdMinLevel: default value is 15. Could be in range from 0 to 15.
- ThresholdCollLevel: default value is 7. Could be in range from 0 to 7.
- RFLevelAmp: default value is 0. On uFR device should be 0 all the time. (1 for **on**, 0 for **off**).
- RxGain: default value is 7. Could be in range from 0 to 7.
- RFLevel: default value is 9. Could be in range from 0 to 15.

SetAdHocEmulationParams

This command set ad-hoc emulation parameters. On uFR power on or reset ad-hoc emulation parameters are set back to their default values.

Function declaration (C language):

```
UFR_STATUS SetAdHocEmulationParams(uint8_t ThresholdMinLevel, uint8_t ThresholdCollLevel, uint8_t RFLevelAmp, uint8_t RxGain, uint8_t RFLevel);
```

- ThresholdMinLevel: default value is 15. Could be in range from 0 to 15.
- ThresholdCollLevel: default value is 7. Could be in range from 0 to 7.
- RFLevelAmp: default value is 0. On uFR device should be 0 all the time. (1 for on, 0 for off).
- RxGain: default value is 7. Could be in range from 0 to 7.
- RFLevel: default value is 9. Could be in range from 0 to 15.

Functions for setting Reader baud rates for ISO14443 – 4A cards

SetSpeedPermanently

Function declaration (C language):

```
unsigned long SetSpeedPermanently(uint8_t tx_speed, uint8_t rx_speed);
```

Parameters description:

- tx_speed – setup value for transmit speed
- rx_speed – setup value for receive speed

Valid speed setup values are:

Const	*Configured speed*
0	106 kbps (default)
1	212 kbps
2	424 kbps

On some reader types maximum rx_speed is 212 kbps. If you try to set higher speed than is allowed, reader firmware will automatically set the maximum possible speed.

Possible error codes:

```
WRITE_VERIFICATION_ERROR = 0x70
```

(command resulting in a direct write to a device non-volatile memory)

GetSpeedParameters

Function declaration (C language):

```
unsigned long GetSpeedParameters(uint8_t* tx_speed, uint8_t* rx_speed);
```

Parameters description:

- tx_speed – returns configured value for transmit speed
- rx_speed – returns configured value for receive speed

FUNCTIONS FOR DISPLAY CONTROL

- **SetDisplayData**

Function enables sending data to the display. A string of data contains information about the intensity of color in each cell of the display. Each cell has three LED (red, green and blue). For each cell of the three bytes is necessary. The first byte indicates the intensity of the green color, the second byte indicates the intensity of the red color, and the third byte indicates the intensity of blue color. For example, if the display has 16 cells, an array contains 48 bytes. Value of intensity is in range from 0 to 255.

Function declaration (C language)

```
unsigned long SetDisplayData(unsigned char  *display_data,
                      unsigned char data_length);
```

Parameters description.

- *display_data* – pointer to data array
- *data_length* – number of data into array

- **SetSpeakerFrequency**

Function sets the frequency of the speaker. The speaker is working on this frequency until a new frequency setting. To stop the operation set frequency to zero.

Function declaration (C language)

```
unsigned long SetSpeakerFrequency(uint16_t frequency);
```

Parameters description

- *frequency* – frequency in Hz

- **SetDisplayIntensity**

Function sets the intensity of light on the display. Value of intensity is in range 0 to 100.

Function declaration (C language)

```
unsigned long SetDisplayIntensity(unsigned char intensity);
```

Parameters description.

- *intensity* – value of intensity (0 – 100)

- **SetDisplayIntensity**

Function gets the intensity of light on the display.

Function declaration (C language)

```
unsigned long GetDisplayIntensity(unsigned char *intensity);
```

Parameters description.

- *intensity* – pointer of value of intensity

FUNCTIONS TO USE THE SHARED RAM INTO DEVICE

Shared RAM is memory space on a device that is used for communication between computer and Android device (phone, tablet) with an NFC reader. PC writes and read data from shared RAM via USB port. Device with Android OS writes and read data from shared RAM via NFC.

- **EnterShareRamCommMode**

Put reader permanently in the mode that use shared RAM. After execution of this function, must be executed function TagEmulationStart.

Function declaration (C language):

```
unsigned long EnterShareRamCommMode(void);
```

- **ExitShareRamCommMode**

The permanent exit from mode that use shared RAM. After execution of this function, must be executed function TagEmulationStop.

Function declaration (C language):

```
unsigned long EnterShareRamCommMode(void);
```

- **WriteShareRam**

Function allows writing data to the shared RAM.

Function declaration (C language):

```
unsigned long WriteShareRam(uint8_t *ram_data,
                            uint8_t addr,
                            uint8_t data_len);
```

Parameters description:

- *ram_data* – pointer to data array
- *addr* – address of first data in an array
- *data_len* – length of array. Address + data_len <= 184

- ***ReadShareRam***

Function allows read data from the shared RAM.

Function declaration (C language)

```
unsigned long ReadShareRam(uint8_t *ram_data,
 uint8_t addr,
 uint8_t data_len);
```

FUNCTIONS FOR DEVICE SIGNALIZATION SETTINGS

- ***GreenLedBlinkingTurnOn***

  The function allows the blinking of the green diode independently of the user's signaling command (default setting).

  Function declaration (C language)

  ```
  unsigned long GreenLedBlinkingTurnOn(void);
  ```

- ***GreenLedBlinkingTurnOff***

  The function prohibits the blinking of the green diode independently of the user's signaling command. LED and sound signaling occurs only on the user command.

  Function declaration (C language)

  ```
  unsigned long GreenLedBlinkingTurnOff(void);
  ```

BASE HD UFR SUPPORT FUNCTIONS

- ***UfrXrcLockOn***

Function description:

Electric strike switches when the function called. Pulse duration determined by function.

Function declaration (C language)

```
unsigned long UfrXrcLockOn(unsigned short pulse_duration);
```

Parameter description:

- pulse_duration is strike switch on period in ms.

- ***UfrXrcRelayState***

Function description:

Function switches relay.

Function declaration (C language)

```
unsigned long UfrXrcRelayState(unsigned char state);
```

Parameter description:

- if the state is 1, then relay is switch on, and if state is 0, then relay is switch off.

- ***UfrXrcGetIoState***

Function description:

Function returns states of 3 IO pins.

Function declaration:

```
unsigned long UfrXrcGetIoState(
unsigned char *intercom,
unsigned char *door,
unsigned char *relay_state);
```

Parameters description:

- intercom shows that there is voltage at the terminals for intercom connection, or not
- door shows that the door's magnetic switch opened or closed.
- relay_state is 1 if relay switch on, and 0 if relay switch off.

FUNCTIONS FOR RF ANALOG REGISTERS SETTING

These functions allow you to adjust the value of several registers on PN512. These are registers: RFCfgReg, RxThresholdReg, GsNOnReg, GsNOffReg, CWGsPReg, ModGsPReg. This can be useful if you want to increase the operation distance of card, or when it is necessary to reduce the impact of environmental disturbances.

- ***SetRfAnalogRegistersTypeA***
- ***SetRfAnalogRegistersTypeB***
- ***SetRfAnalogRegistersISO14443_212***
- ***SetRfAnalogRegistersISO14443_424***

Functions description:

Functions allow adjusting values of registers RFCfgReg and RxThresholdReg. Registry setting is applied to the appropriate type of communication with tag. There are ISO14443 Type A, ISO14443 TypeB, and ISO14443-4 on higher communication speeds (211 and 424 Kbps).

Functions declaration (C language):

```
unsigned long SetRfAnalogRegistersTypeA(uint8_t ThresholdMinLevel,
                      uint8_t ThresholdCollLevel,uint8_t RFLevelAmp,
                      uint8_t RxGain, uint8_t RFLevel);

unsigned long SetRfAnalogRegistersTypeB(uint8_t ThresholdMinLevel,
                      uint8_t ThresholdCollLevel,uint8_t RFLevelAmp,
                      uint8_t RxGain, uint8_t RFLevel);

unsigned long SetRfAnalogRegistersISO14443_212(
                      uint8_t ThresholdMinLevel,
                      uint8_t ThresholdCollLevel,uint8_t RFLevelAmp,
                      uint8_t RxGain, uint8_t RFLevel);

unsigned long SetRfAnalogRegistersISO14443_424(
                      uint8_t ThresholdMinLevel,
                      uint8_t ThresholdCollLevel,uint8_t RFLevelAmp,
                      uint8_t RxGain, uint8_t RFLevel);
```

Parameters description:

- TheshodMinLevel (0 – 15) and ThresholCollLevel (0 – 7) are parts of RxThresholdReg
- RFLevelAmp (0 or 1), RxGain (0 – 7) and RFLevel (0 – 15) are parts of RFCfgReg .

- ***SetRfAnalogRegistersTypeADefault***
- ***SetRfAnalogRegistersTypeBDefault***
- ***SetRfAnalogRegistersISO14443_212Default***
- ***SetRfAnalogRegistersISO14443_424Default***

Functions description:

The functions set the factory default settings of the registers RFCfgReg and RxThresholdReg.

Functions declaration (C language):

```
unsigned long SetRfAnalogRegistersTypeADefault(void);

unsigned long SetRfAnalogRegistersTypeBDefault(void);

unsigned long SetRfAnalogRegistersISO14443_212Default(void);

unsigned long SetRfAnalogRegistersISO14443_424Default(void);
```

- ***GetRfAnalogRegistersTypeA***
- ***GetRfAnalogRegistersTypeB***
- ***GetRfAnalogRegistersISO14443_212***
- ***GetRfAnalogRegistersISO14443_424***

Functions description:

The functions read the value of the registers RFCfgReg and RxThresholdReg.

Functions declaration (C language):

```
unsigned long GetRfAnalogRegistersTypeA(uint8_t *ThresholdMinLevel,
                      uint8_t *ThresholdCollLevel, uint8_t *RFLevelAmp,
                      uint8_t *RxGain, uint8_t *RFLevel);

unsigned long GetRfAnalogRegistersTypeB(uint8_t *ThresholdMinLevel,
                      uint8_t *ThresholdCollLevel, uint8_t *RFLevelAmp,
                      uint8_t *RxGain, uint8_t *RFLevel);

unsigned long GetRfAnalogRegistersISO14443_212(
                      uint8_t *ThresholdMinLevel,
                      uint8_t *ThresholdCollLevel, uint8_t *RFLevelAmp,
                      uint8_t *RxGain, uint8_t *RFLevel);

unsigned long GetRfAnalogRegistersISO14443_424(
                      uint8_t *ThresholdMinLevel,
                      uint8_t *ThresholdCollLevel, uint8_t *RFLevelAmp,
                      uint8_t *RxGain, uint8_t *RFLevel);
```

Parameters description:

- TheshodMinLevel (0 – 15) and ThresholCollLevel (0 – 7) are parts of RxThresholdReg
- RFLevelAmp (0 or 1), RxGain (0 – 7) and RFLevel (0 – 15) are parts of RFCfgReg.

- ***SetRfAnalogRegistersTypeATrans***
- ***SetRfAnalogRegistersTypeBTrans***

Functions allow adjusting values of registers RFCfgReg, RxThresholdReg, GsNOnReg, GsNOffReg, CWGsPReg, ModGsPReg. Registry setting is applied to the appropriate type of communication with tag. There are ISO14443 Type A, ISO14443 TypeB, and ISO14443-4 on higher communication speeds (211 and 424 Kbps).

Functions declaration (C language):

```
unsigned long SetRfAnalogRegistersTypeATrans(
                      uint8_t ThresholdMinLevel,
                      uint8_t ThresholdCollLevel,uint8_t RFLevelAmp,
                      uint8_t RxGain, uint8_t RFLevel,
                      uint8_t CWGsNOn,uint8_t ModGsNOn, uint8_t CWGsP,
                      uint8_t CWGsNOff, uint8_t ModGsNOff);

unsigned long SetRfAnalogRegistersTypeBTrans(
                      uint8_t ThresholdMinLevel,
                      uint8_t ThresholdCollLevel, uint8_t RFLevelAmp,
                      uint8_t RxGain, uint8_t RFLevel,
                      uint8_t CWGsNOn, uint8_t ModGsNOn, uint8_t CWGsP,
                      uint8_t ModGsP);
```

Parameters description:

- TheshodMinLevel (0 – 15) and ThresholCollLevel (0 – 7) are parts of RxThresholdReg
- RFLevelAmp (0 or 1), RxGain (0 – 7) and RFLevel (0 – 15) are parts of RFCfgReg.
- CWGsNOn (0 – 15) and ModGsNOn (0 – 15) are parts of GsNOnReg
- CWGsP (0 – 47) is value of CWGsPReg
- CWGsNOff (0 – 15) and ModGsNOff (0 – 15) are parts of GsNOffReg
- ModGsP (0 – 47) is value of ModGsPReg

- ***SetRfAnalogRegistersTypeATrans***
- ***SetRfAnalogRegistersTypeBTrans***

Functions description:

The functions read the value of the registers RFCfgReg, RxThresholdReg, GsNOnReg, GsNOffReg, CWGsPReg, ModGsPReg.

Functions declaration (C language):

```
unsigned long GetRfAnalogRegistersTypeATrans(
                        uint8_t *ThresholdMinLevel,
                        uint8_t *ThresholdCollLevel, uint8_t *RFLevelAmp,
                        uint8_t *RxGain, uint8_t *RFLevel,
                        uint8_t *CWGsNOn, uint8_t *ModGsNOn,
                        uint8_t *CWGsP, uint8_t *CWGsNOff,
                        uint8_t *ModGsNOff);

unsigned long GetRfAnalogRegistersTypeBTrans(
                        uint8_t *ThresholdMinLevel,
                        uint8_t *ThresholdCollLevel, uint8_t *RFLevelAmp,
                        uint8_t *RxGain, uint8_t *RFLevel,
                        uint8_t *CWGsNOn, uint8_t *ModGsNOn,
                        uint8_t *CWGsP, uint8_t *ModGsP);
```

Parameters description:

- TheshodMinLevel (0 – 15) and ThresholCollLevel (0 – 7) are parts of RxThresholdReg
- RFLevelAmp (0 or 1), RxGain (0 – 7) and RFLevel (0 – 15) are parts of RFCfgReg.
- CWGsNOn (0 – 15) and ModGsNOn (0 – 15) are parts of GsNOnReg
- CWGsP (0 – 47) is value of CWGsPReg
- CWGsNOff (0 – 15) and ModGsNOff (0 – 15) are parts of GsNOffReg
- ModGsP (0 – 47) is value of ModGsPReg

Change log:

date	page	description	refers to the lib version / firmware ver.
1.12.2017.	85	Functions for RF analog registers setting	4.3.3/3.9.54
7.11.2017.	83	Base HD uFR support functions	BaseHD_FW_uFRsupport
6.11.2017.	82	Functions for device signalization settings	4.3.2/3.9.53
28.07.2017.	59	Functions supporting work with Desfire Value files	4.0.28 /3.9.43
29.06.2017.	68	Fully uFR firmware support for ISO 14443-4A protocol commands	4.0.26 / 3.9.39
23.05.2017.	65	Support for ISO 14443-4A protocol commands	4.0.25 / 3.9.36
23.05.2017.	66	Support for APDU commands in ISO 14443-4A tags update.	4.0.25 / 3.9.36
03.05.2017.	77	Functions supporting Ad-Hoc emulation mode parameters manipulation. (GetAdHocEmulationParams() and SetAdHocEmulationParams()).	4.0.24 / 3.9.35
03.05.2017.	77	Functions supporting Ad-Hoc emulation mode.	4.0.23 / 3.9.34
31.08.2016.	67	Functions for display control updated	4.0.10 / 3.7.0
31.08.2016.	66	(DEPRECATED – replaced by “Support for APDU commands in ISO 14443-4 tags update” functionality from 23.05.2017) Support for APDU commands in ISO 14443-4 tags.	4.0.10 / 3.7.0
06.08.2016.	7	FAST_READ ISO14443-3 command with LinearRead utilisation.	any lib ver / 3.9.14

06.06.2016.	7, 9, 12, 14, 73.	*ucAuthMode* parameter description changed regarding authentication mode considerations for NTAG 21x and other T2T tags. Applied to lib functions: *LinearRead***,** *LinearWrite, BlockRead, BlockWrite and LinearRowRead.*	any lib ver / 3.9.10